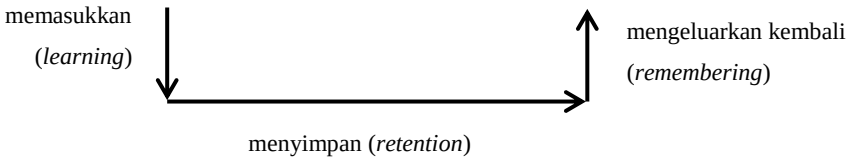
BAB 2
LANDASAN TEORI

2.1 Ingatan

Ingatan memberikan bermacam – macam arti bagi para ahli. Pada umumnya para ahli memandang ingatan sebagai sebuah hubungan antara pengalaman dengan masa lampau. Dengan adanya kemampuan mengingat pada manusia, hal ini menunjukkan bahwa manusia mampu menerima, menyimpan, dan menimbulkan kembali pengalaman – pengalaman yang dialaminya.

Telah dikemukakan di atas bahwa ingatan berhubungan dengan pengalaman – pengalaman yang telah lampau. Dengan demikian dapatlah dikemukakan bahwa apa yang diingat merupakan hal yang telah pernah dialami atau pernah dipersepsinya. Dengan demikian apabila ditinjau lebih lanjut, ingatan itu tidak hanya kemampuan untuk menyimpan apa yang telah pernah dialaminya saja, tetapi juga meliputi kemampuan untuk menerima, menyimpan, dan menimbulkan kembali. Secara skematis dapat dikemukakan bahwa ingatan itu mencakup kemampuan – kemampuan sebagai berikut:



Gambar 2.1 Skema Ingatan [1]

Dari skema diatas dapat dikemukakan bahwa ingatan merupakan kemampuan psikis untuk memasukkan (*learning*), menyimpan (*retention*), dan menimbulkan kembali (*remembering*) hal – hal yang lampau. Istilah lain yang juga sering digunakan yaitu memasukkan (*encoding*), menyimpan (*storage*), dan menimbulkan kembali (*retrieval*) [1].

Ingatan atau sering juga disebut dengan memori dibagi menjadi dua, yaitu memori jangka pendek (*Short-Term Memory* / STM) dan memori jangka panjang (*Long-Term Memory* / LTM). Informasi sebelum disimpan di memori jangka

panjang, diproses dahulu di memori jangka pendek. STM disebut juga dengan istilah memori kerja (*working memory*). Memori kerja didefinisikan secara konseptual sebagai tipe meja kerja (*workbench*) yang secara konstan mengubah, mengkombinasikan, dan memperbarui informasi baru dan lama. Model memori kerja menyanggah pandangan bahwa STM hanyalah sekedar suatu "kotak" di kepala, semacam unit pemrosesan sederhana tempat informasi dikirim ke LTM atau lenyap [5].

2.2 Perkembangan Kognitif

Kognitif adalah bagaimana seseorang beradaptasi dan menginterpretasikan objek, keadaan dirinya, dan kejadian-kejadian di sekitarnya. Secara umum, kognitif diartikan sebagai potensi intelektual yang terdiri dari tahapan: pengetahuan, pemahaman, penerapan, analisa, sintesa, dan evaluasi. Menurut kamus besar bahasa Indonesia, kognitif adalah berhubungan dengan atau melibatkan kognisi, berdasarkan kepada pengetahuan faktual yang empiris.

Kognitif berarti persoalan yang menyangkut kemampuan untuk mengembangkan kemampuan rasional (akal). Teori kognitif lebih menekankan bagaimana proses atau upaya untuk mengoptimalkan kemampuan aspek rasional yang dimiliki oleh orang lain. Oleh sebab itu, kognitif berbeda dengan teori behavioristik yang lebih menekankan pada aspek kemampuan perilaku yang diwujudkan dengan cara kemampuan merespons terhadap stimulus yang datang kepada dirinya.

Perkembangan kognitif seseorang tidak hanya ditentukan dari pertumbuhan dan kematangan sistem saraf pusat maupun perifer saja, namun juga cara ia memproses informasi, meningkatkan daya ingat dan kapasitas memorinya, dan kedekatannya dengan suatu objek pengetahuan. Walaupun demikian, tingkat kematangan kognitif seseorang dapat ditingkatkan dengan latihan-latihan dan usaha untuk memperbaiki cara belajar dan mengorganisasi memori. Hal ini juga tidak terlepas dari potensi-potensi yang dimilikinya, termasuk bakat tentang pengetahuan tertentu. Suatu hal yang harus diperhatikan pada perkembangan

kognitif adalah bukan pada cara berfikir dan banyaknya informasi yang dikuasainya, namun lebih kepada cara menggunakan informasi yang dimilikinya untuk menyelesaikan permasalahan yang dihadapi. Pertumbuhan otak mencapai kesempurnaan pada usia 12–25 tahun secara fungsional, perkembangan kognitif (kemampuan berfikir) pada rentang usia tersebut, secara intelektual seseorang mulai dapat berfikir logis tentang suatu informasi tidak hanya diingat melainkan diproses lebih mendalam [13].

2.3 Metode N-Back

Latihan n-back adalah latihan kinerja berkesinambungan yang umum digunakan untuk melatih memori kerja (*working memory*) [4]. Metode ini pertama kali diperkenalkan oleh Wayne Kirchner pada tahun 1958 [14]. Latihan n-back adalah sebuah latihan yang menampilkan himpunan terbatas *item* yang dapat berupa angka, huruf, maupun gambar dalam aliran berkelanjutan, dimana subjek diharuskan untuk menanggapi *item* tersebut yang sesuai dengan *item* yang pada n posisi sebelumnya.

Item yang ditampilkan biasanya berbentuk visual atau auditori. *Item* visual adalah karakteristik dari latihan n-back yang mencakup huruf, kata, dan angka ditampilkan secara bergantian dengan *delay* tertentu. Sedangkan, *item* auditori berbentuk suara. Pada saat pengerjaan latihan n-back ini, subjek disajikan dengan urutan *item* dapat berupa angka ataupun huruf dan subjek memilih *item* ketika *item* saat ini cocok dengan salah satu dari n langkah awal dalam urutan. Jumlah n dapat diatur untuk menentukan kesulitan.

Latihan dengan metode n-back ini berusaha untuk menghasilkan peningkatan kapasitas memori kerja dengan berfokus pada penguatan proses pada memori kerja. Selain itu, terdapat lima efek yang dapat diperoleh dari hasil latihan n-back, yaitu peningkatan pada perhatian khusus, peningkatan pada pembaharuan informasi, peningkatan pada pergantian fokus, dan peningkatan pada fokus perhatian. Dapat disimpulkan bahwa tujuan utama dari latihan n-back adalah untuk mempengaruhi kemudahan dan keberhasilan kemampuan kognitif, seperti

peningkatan kemampuan memori kerja yang berkaitan dengan kemampuan daya ingat seseorang dalam kehidupan sehari-hari bukan hanya saat latihan saja. Ketika pelatihan diimplementasikan pada individu dengan gangguan mental yang spesifik, tujuan pelatihan adalah pengentasan gejala tertentu dari gangguan [2].

Ketika n sama dengan 2 atau lebih, tidak cukup hanya menyimpan *item* yang baru, memori kerja perlu diperbarui terus menerus untuk melacak apakah *item* saat ini cocok atau tidak dengan n *item* sebelumnya. Tugas n-back mengaktifkan memori kerja pada saat pengerjaannya [4]. Pada saat seseorang mengerjakan tugas n-back, beberapa bagian otak aktif secara konsisten, diantaranya yang berkaitan dengan memori kerja [15].

Pada latihan n-back yang telah terkomputerisasi, interval antar *item* selama 2,5 detik. Subjek diuji dengan 2-back sampai 4-back. Untuk tiap latihan terdapat 3 set dan untuk tiap set berjumlah 20 + n *item* dengan 6 jawaban yang benar, dengan begitu rumus dari jumlah *item* untuk tiap latihan, yaitu (20 + n) x 3.



Gambar 2.2 Contoh 2-back Huruf

Metode n-back sekarang banyak digunakan untuk penelitian, klinis, dan juga medis. Sebuah lembaga swasta memanfaatkan metode n-back (dalam hubungannya dengan tugas-tugas kognitif lainnya) untuk meningkatkan kecerdasan klien mereka. Perusahaan bimbingan dan psikolog juga memanfaatkan metode n-back ini untuk meningkatkan fokus individu [16] dan untuk merehabilitasi penderita cedera otak traumatis [17], percobaan telah menemukan bukti bahwa berlatih dengan metode n-back ini dapat membantu orang – orang menjadi lebih fokus setelah pelatihan.

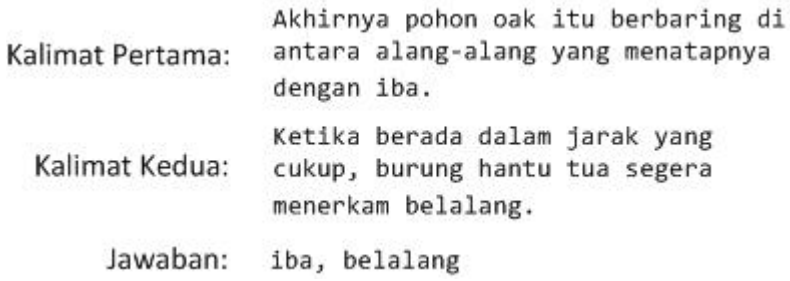
2.4 Pengujian Kemampuan Daya Ingat

Istilah memori kerja dikembangkan sebagai cara untuk merujuk ke bagian yang lebih aktif dari sistem pengolahan manusia. Memori kerja diasumsikan memiliki pengolahan serta fungsi penyimpanan, memori ini berfungsi sebagai tempat untuk mengeksekusi proses dan untuk menyimpan hasil dari proses tersebut. Informasi dapat menjadi bagian dari memori kerja melalui beberapa rute, yaitu berdasarkan perseptual yang dikodekan dari teks, diambil dari memori jangka panjang , dan *output* dari proses pemahaman.

Salah satu cara untuk mengukur kemampuan memori kerja adalah dengan menggunakan *Reading Span Test* (RST). Tes ini diperkenalkan pertama kali oleh Meredyth Daneman dan Patricia A. Carpenter. Tes ini dirancang untuk mengukur fungsi pengolahan dan penyimpanan pada memori kerja. Subjek diberikan kalimat untuk dibaca, pada akhir set subjek mencoba untuk mengingat kata akhir dari setiap kalimat. Jumlah kalimat dalam satu set itu bertambah dari percobaan ke percobaan dan rentang membaca subjek adalah jumlah maksimum kalimat ia bisa membaca sambil mempertahankan pemanggilan kembali (*recall*) sempurna dari kata-kata terakhir dari tiap kalimat.

Reading Span Test (RST) dibangun sehingga tuntutan yang sesuai dengan karakteristik dari memori kerja. Pertama, subjek ditampilkan satu kalimat selama sesuai dengan kemampuan membaca subjek. Segera setelah satu kalimat selesai ditampilkan, kemudian disajikan kalimat berikutnya sehingga subjek tidak dapat membaca ulang kata-kata dan begitu seterusnya sesuai dengan jumlah kalimat yang akan ditampilkan. Kedua, jumlah informasi yang subjek harus pertahankan cocok dengan kapasitas yang dikaitkan dengan memori kerja. Misalnya, saat membaca kalimat ketiga dalam satu set, subjek harus memiliki pengolahan dan kapasitas penyimpanan yang cukup untuk kalimat tersebut, sementara tetap mempertahankan beberapa kata yang akan memungkinkan subjek untuk mengambil kata-kata terakhir dari dua kalimat sebelumnya.

Dalam ekperimen yang dilakukan oleh Daneman dan Carpenter, tes ini menggunakan 20 subjek penelitian. Setiap subjek harus membaca 60 kalimat yang tidak saling berkaitan, dengan 13 sampai 16 huruf per kalimat. Setiap kalimat diakhiri dengan kata yang berbeda.



Gambar 2.3 Contoh *Reading Span Test*

Pada contoh dua kalimat tersebut memiliki dua kata terakhir yang berbeda. Pada tes yang diimplementasikan, 60 kalimat yang digunakan dibagi menjadi 5 bagian, yaitu 2 kalimat, 3 kalimat, 4 kalimat, 5 kalimat, dan 6 kalimat. Pada setiap bagian tersebut masing – masing memiliki 3 set. Setiap subjek diberikan penjelasan (*tutorial*) sebelum memulai tes tersebut [18].

Skoring yang digunakan pada *Reading Span Test* (RST) adalah dengan persentase kata yang diingat, maksudnya yaitu persentase kata yang diingat dari tiap bagian tes. Nantinya skor akan disajikan untuk tiap bagian. Total skor maksimum yang dapat diperoleh untuk tiap bagian adalah 100% [19].

2.5 Android

Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware*, dan aplikasi. Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka.

Tidak hanya menjadi sistem operasi di *smartphone*, saat ini Android menjadi pesaing utama Apple pada sistem operasi Tablet PC. Pesatnya pertumbuhan Android adalah karena Android itu sendiri adalah *platform* yang sangat lengkap baik itu sistem operasinya, aplikasi dan *tool* pengembangan,

market aplikasi Android serta dukungan yang sangat tinggi dari komunitas *open source* di dunia, sehingga Android terus berkembang pesat baik dari segi teknologi maupun dari segi jumlah *device* yang ada di dunia.

Android merupakan generasi baru *platform mobile*, *platform* yang memberikan pengembang untuk melakukan pengembangan sesuai dengan yang diharapkannya. Komersialisasi pengembang (produsen *handset* khususnya) dapat memilih untuk meningkatkan *platform* tanpa harus memberikan perbaikan mereka ke masyarakat *open source*. Pengembang aplikasi Android diperbolehkan untuk mendistribusikan aplikasi mereka di bawah skema lisensi apapun yang mereka inginkan. Adapun beberapa versi Android sampai sekarang, yaitu:

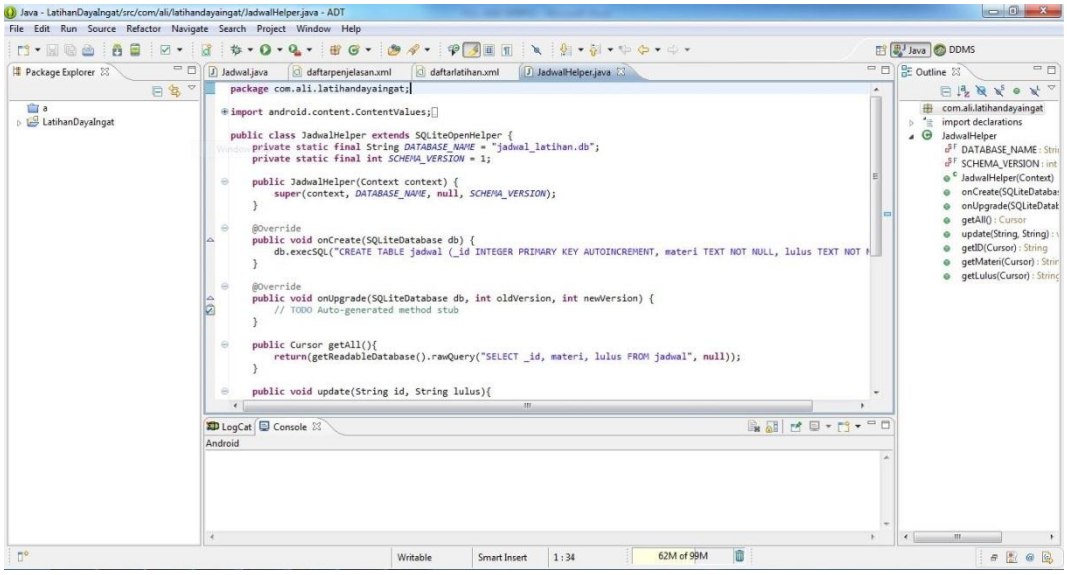
1. Android 1.0 (API Level 1).
2. Android 1.1 (API Level 2).
3. Android 1.5 *Cupcake* (API Level 3).
4. Android 1.6 *Donut* (API Level 4).
5. Android 2.0 *Eclair* (API Level 5).
6. Android 2.0.1 *Eclair* (API Level 6).
7. Android 2.1 *Eclair* (API Level 7).
8. Android 2.2-2.2.3 *Froyo* (API Level 8).
9. Android 2.3-2.3.2 *Gingerbread* (API Level 9).
10. Android 2.3.3-2.3.7 *Gingerbread* (API Level 10).
11. Android 3.0 *Honeycomb* (API Level 11).
12. Android 3.1 *Honeycomb* (API Level 12).
13. Android 3.2 *Honeycomb* (API Level 13).
14. Android 4.0-4.0.2 *Ice Cream Sandwich* (API Level 14).
15. Android 4.0.3-4.0.4 *Ice Cream Sandwich* (API Level 15).
16. Android 4.1 *Jelly Bean* (API Level 16).
17. Android 4.2 *Jelly Bean* (API Level 17).
18. Android 4.3 *Jelly Bean* (API Level 18).
19. Android 4.4 *KitKat* (API Level 19).

2.5.1 Android SDK (*Software Development Kit*)

Android SDK adalah *tools* API (*Application Programming Interface*) yang diperlukan untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman Java. Saat ini disediakan Android SDK (*Software Development Kit*) sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman Java. Sebagai *platform* aplikasi-netral, Android memberi kesempatan untuk membuat aplikasi yang dibutuhkan [20].

2.5.2 Eclipse

Pengembang memiliki beberapa pilihan ketika membuat aplikasi yang berbasis Android. Sebagian besar pengembang menggunakan *Eclipse* yang tersedia secara bebas untuk merancang dan mengembangkan aplikasi Android. *Eclipse* adalah IDE yang paling populer untuk pengembangan Android, karena memiliki Android *plug-in* yang tersedia untuk memfasilitasi pengembangan Android.



Gambar 2.4 Antarmuka Eclipse

Selain itu, *Eclipse* juga mendapat dukungan langsung dari Google untuk menjadi IDE pengembangan aplikasi Android, ini terbukti dengan adanya penambahan *plugins* untuk *Eclipse* untuk membuat *project* Android dimana

source software langsung dari situs resminya Google [20]. Akan tetapi, tidak menutup kemungkinan untuk menggunakan IDE yang lain, seperti *Netbeans*.

2.5.3 ADT (*Android Development Tools*)

ADT (*Android Development Tools*) adalah *plugin* yang didesain untuk IDE *Eclipse* yang memberikan kemudahan dalam mengembangkan aplikasi Android dengan menggunakan IDE *Eclipse*. Dengan menggunakan ADT untuk *Eclipse* akan memudahkan pengembang aplikasi dalam membuat *project* Android, membuat GUI aplikasi, dan menambahkan komponen – komponen yang lainnya, begitu juga dapat melakukan *running* aplikasi menggunakan Android SDK melalui *Eclipse*. Dengan ADT pengembang juga dapat melakukan pembuatan *package* Android (.apk) yang digunakan untuk distribusi aplikasi yang telah dibuat [20].

2.5.4 Arsitektur Android

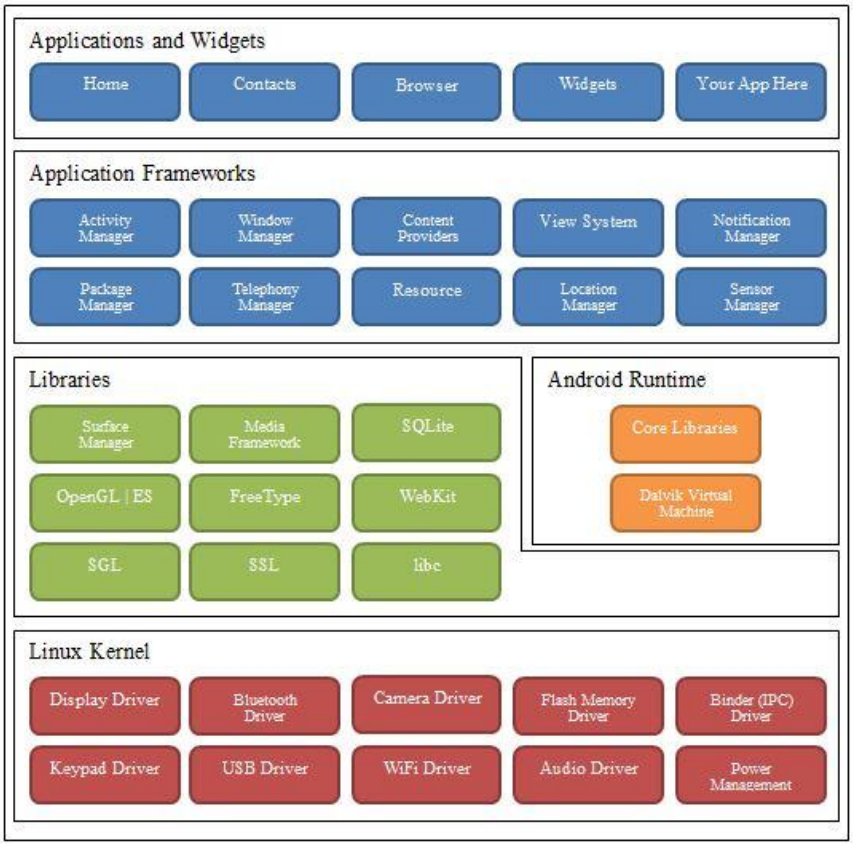
Secara garis besar arsitektur Android dapat dijelaskan dan digambarkan sebagai berikut [20]:

1. *Applications and Widgets*

Applications and Widgets adalah *layer* dimana pengguna berhubungan dengan aplikasi saja, dimana biasanya pengguna mengunduh aplikasi kemudian pengguna melakukan instalasi dan menjalankan aplikasi tersebut. Di *layer* ini terdapat aplikasi inti termasuk email, SMS, kalender, peta, *browser*, kontak, dan lain sebagainya. Semua aplikasi ditulis menggunakan bahasa pemrograman Java.

2. *Application Frameworks*

Android adalah *Open Development Platform* yaitu menawarkan kepada pengembang atau memberi kemampuan kepada pengembang untuk membangun aplikasi yang bagus dan inovatif. Pengembang bebas untuk mengakses perangkat keras, akses informasi *resources*, menjalankan *service background*, mengatur *alarm*, dan memiliki akses penuh menuju API *framework* seperti yang dilakukan oleh aplikasi yang kategori inti. Arsitektur aplikasi dirancang supaya dengan mudah dapat menggunakan kembali komponen yang sudah digunakan (*reuse*).



Gambar 2.5 Arsitektur Android [20]

Sehingga bisa disimpulkan *Application Frameworks* ini adalah *layer* dimana para pembuat aplikasi melakukan pengembangan / pembuatan aplikasi yang akan dijalankan pada sistem operasi Android., karena pada *layer* inilah aplikasi dapat dirancang dan juga dibuat.

3. *Libraries*

Libraries ini adalah *layer* dimana fitur – fitur Android berada, biasanya para pembuat aplikasi mengakses *libraries* untuk menjalankan aplikasinya. Pada *layer* ini meliputi berbagai *library*, seperti:

a. *Libraries* media untuk pemutaran media audio dan video.
b. *Libraries* untuk manajemen tampilan.
c. *Libraries Graphics* mencakup SGL dan OpenGL.
d. *Libraries* SQLite untuk dukungan *database*.
e. *Libraries* SSL dan *WebKit* terintegrasi dengan *web browser* dan *security*.

f. *Libraries* LiveWebCore mencakup modern *web browser* dengan *engine embedded web view*.
g. *Libraries* 3D yang mencakup implementasi OpenGL ES 1.0 API's.

4. Android *Runtime*

Layer yang membuat aplikasi Android dapat dijalankan dimana dalam prosesnya menggunakan implementasi Linux. *Dalvik Virtual Machine* (DVM) merupakan mesin yang membentuk dasar kerangka aplikasi Android. Di dalam Android *Runtime* dibagi menjadi dua bagian, yaitu:

a. *Core Libraries*: Aplikasi Android dibangun dalam bahasa Java, sementara Dalvik sebagai virtual mesinnya, sehingga diperlukan *libraries* yang digunakan untuk menterjemahkan bahasa Java yang ditangani oleh *Core Libaries*.
b. *Dalvik Virtual Machine*: Virtual mesin yang dioptimalkan untuk menjalankan fungsi secara efisien, dimana merupakan pengembangan yang mampu membuat Linux *Kernel* untuk melakukan *threading*.

5. Linux *Kernel*

Linux *Kernel* adalah *layer* dimana inti dari sistem operasi Android berada. Berisi *file – file system* yang mengatur sistem pemrosesan, memori, *resource*, *drivers*, dan lain sebagainya.

2.5.5 Komponen Aplikasi Android

Aplikasi Android ditulis dalam bahasa pemrograman Java. Kode Java dikompilasi bersama dengan data *file resource* yang dibutuhkan oleh aplikasi, dimana prosesnya di-*package* oleh *tools* yang dinamakan "apt tools" ke dalam paket Android sehingga menghasilkan *file* dengan ekstensi apk. *File* apk tersebut yang nantinya dapat dipasang di perangkat Android. Adapun enam jenis komponen pada aplikasi Android, yaitu sebagai berikut [20]:

1. *Activities*

Suatu *activity* akan menyajikan *user interface* (UI) kepada pengguna, sehingga pengguna dapat melakukan interaksi. Sebuah aplikasi Android bisa jadi hanya

memiliki satu *activity*, tetapi umumnya aplikasi memiliki banyak *activity* tergantung pada tujuan aplikasi dan desain dari aplikasi tersebut. Satu *activity* biasanya akan dipakai untuk menampilkan aplikasi atau yang bertindak sebagai *user interface* (UI) saat aplikasi diperlihatkan kepada *user*. Untuk pindah dari satu *actvity* ke *activity* yang lain dapat melakukannya dengan satu *event*, misalnya dengan tombol. Secara hirarki sebuah *windows activity* dinyatakan dengan *method* Activity.setContentView(). ContentView adalah objek yang berada pada *root* hirarki.

2. *Service*

Service tidak memiliki *Graphic User Interface* (GUI), tetapi *service* berjalan secara *background*, contoh dalam memainkan musik, *service* mungkin memainkan musik atau mengambil data dari jaringan, tetapi setiap *service* harus berada dalam kelas induknya. Misalnya, *media player* sedang memutar lagu dari *list* yang ada, aplikasi ini akan memiliki dua atau lebih *activity* yang memungkinkan *user* untuk memilih lagu misalnya, atau menulis sms sambil *player* jalan. Untuk menjaga musik tetap dijalankan, *activity player* dapat menjalalankan *service*. *Service* dijalankan pada *thread* utama dari proses aplikasi.

3. *Broadcast Receiver*

Broadcast receiver berfungsi menerima dan bereaksi untuk menyampaikan notifikasi. Contoh *broadcast* seperti notifikasi zona waktu berubah, bateri *low*, gambar telah selesai diambil dari kamera, atau pengubahan referensi bahasa yang digunakan. *Broadcast receiver* tidak memiliki *user interface* (UI), tetapi memiliki sebuah *activity* untuk merespon informasi yang mereka terima, atau mungkin menggunakan *notification manager* untuk memberitahu *user*, seperti lampu latar atau getaran perangkat, dan lain sebagainya.

4. *Content Provider*

Content provider membuat kumpulan aplikasi data secara spesifik sehingga dapat digunakan oleh aplikasi lain. Data disimpan dalam *file* sistem seperti *database* SQLite. *Content provider* menyediakan cara untuk mengakses data yang

dibutuhkan oleh suatu *activity*, misalnya ketika akan menggunakan aplikasi yang membutuhkan peta (*Map*), atau aplikasi yang membutuhkan untuk mengakses data kontak dan navigasi, maka disinilah fungsi dari *content provider*.

2.5.6 Bahasa Pemrograman Java

Bahasa pemrograman Java adalah bahasa yang bersifat umum (*general-purpose*), berbasis kelas dan juga berbasis objek. Bahasa pemrograman Java ini dirancang untuk menjadi cukup sederhana sehingga banyak *programmer* dapat mengerti bahasa ini. Bahasa pemrograman Java ini berkaitan dengan C dan C++ tetapi dengan organisasi yang agak berbeda, dengan sejumlah aspek C dan C++ dihilangkan dan beberapa ide dari bahasa lain disertakan.

Java merupakan bahasa yang *strongly typed*, karena setiap variabel harus dideklarasikan dengan tipe data. Spesifikasi ini jelas membedakan antara kesalahan saat kompilasi yang dapat dan harus dideteksi pada waktu kompilasi, dan yang terjadi pada *run time*. Waktu kompilasi biasanya terdiri dari menerjemahkan program menjadi representasi *byte code*. Kegiatan *runtime* termasuk *loading* dan *lingking* kelas – kelas yang dibutuhkan untuk mengeksekusi program, *optional machine code generation* dan *dynamic optimization* program, dan eksekusi program yang sebenarnya [21].

2.5.7 SQLite

Android memiliki fasilitas untuk membuat *database* yang dikenal dengan SQLite. SQLite adalah salah satu *software* yang sangat populer, kobinasi SQL *interface* dan penggunaan memori yang sangat sedikit dengan kecepatan yang sangat cepat. SQLite di Android termasuk dalam Android *runtime*, sehingga setiap versi dari Android dapat membuat *database* dengan SQLite.

Tidak ada *database* yang otomatis disediakan oleh Android, jika menggunakan SQLite, harus membuat *database* sendiri, mendefinisikan tabelnya, *index* serta datanya. Banyak *software manager* SQLite yang bisa diunduh secara gratis, salah satunya adalah SQLite yang *addons* dengan Mozilla Firefox. SQLite Manager ini nantinya digunakan untuk melihat data di *database* [20]. Berikut

adalah implementasi *Create, Read, Update, Delete* (CRUD) pada SQLite dengan menggunakan bahasa pemrograman Java:

1. *Create*

```
db.insert(NAMA_TABEL, null, values);
```

2. *Read*

```
String query = "SELECT  * FROM " + NAMA_TABEL;
SQLiteDatabase db = this.getWritableDatabase();
Cursor cursor = db.rawQuery(query, null);
```

3. *Update*

```
db.update(NAMA_TABEL, values, KEY_ID+" = ?", new String[] {
String.valueOf(Object.getId()) });
```

4. *Delete*

```
db.delete(NAMA_TABEL,  KEY_ID+"  =  ?",  new  String[]  {
String.valueOf(Object.getId()) });
```

2.6 Analisis dan Perancangan Berorientasi Objek

Analisis dan perancangan berorientasi objek menganalogikan sistem aplikasi seperti kehidupan nyata yang didominasi oleh objek. Di dalam membangun sistem berorientasi objek akan menjadi lebih baik apabila langkah awalnya didahului dengan proses analisis dan perancangan yang berorientasi objek. Tujuannya adalah mempermudah *programmer* didalam mendesain program dalam bentuk objek-objek dan hubungan antar objek tersebut untuk kemudian dimodelkan dalam sistem nyata.

Unified Modelling Language (UML) adalah bahasa standar dalam mempresentasikan analisis dan perancangan berorientasi objek. Bahasa UML memberikan banyak fitur opsional dalam mempresentasikan semua aspek penting dalam sebuah sistem. Standar yang digunakan sekarang adalah dengan menggunakan UML 2.0. Dalam UML 2.0 terdapat 13 diagram berbeda untuk digunakan dalam memodelkan perangkat lunak [22], diantaranya yaitu *use case diagram, activity diagram, sequence diagram, class diagram, component diagram,* dan *deployment diagram*.

2.6.1 *Use Case Diagram*

Use case diagram membantu dalam menentukan fungsi dan fitur dari perangkat lunak dari perspektif pengguna. Sebuah *use case* mendeskrpskan bagaimana *user* berinteraksi dengan sistem dengan mendefinisikan langkah – langkah yang dibutuhkan untuk mencapai tujuan yang spesifik [22]. Jika *use case* adalah sebuah kebutuhan, maka harus jelas kriteria gagal atau berhasilnya. Pengembang, penguji, dan pengguna secara eksplisit harus tahu apakah suatu sistem memiliki *use case* tersebut atau tidak [23].

Sebuah diagram UML *use case* merupakan gambaran dari semua *use case* dan bagaimana mereka terkait. Dalam diagram *use case*, *use case* ditampilkan sebagai oval. Para aktor dihubungkan oleh garis kepada *use case* yang berhubungan. Perhatikan bahwa tidak ada rincian *use case* pada diagram dan harus dijelaskan secara terpisah. Perhatikan juga bahwa *use case* ditempatkan dalam persegi panjang tetapi aktor tidak. Persegi panjang ini adalah pengingat visual dari batas-batas sistem dan bahwa pelaku berada di luar sistem.



Gambar 2.6 Contoh *Use Case CMS* [23]

Dalam *use case diagram* tersebut belum menjelaskan mendetail mengenai *use case* yang tertera. Untuk menjelaskan *use case* tersebut memerlukan skenario *use case*. Skenario *use case* mendeskripsikan secara detail tentang informasi dari suatu *use case* yang terdapat pada *use case diagram*. Berikut adalah tipe – tipe informasi yang digunakan dalam skenario *use case* [23]:

Tabel 2.1 Tipe Informasi pada Skenario *Use Case* [23]

Use case name	Memberikan penjelasan singkat tentang nama dari *use case*.
Related Requirements	Daftar *use case* yang berhubungan dengan use case tersebut.
Goal In Context	Menjelaskan apa yang aktor coba untuk dapatkan dari *use case*.
Preconditions	Kondisi sistem sebelum *use case* dijalankan.
Successful End Condition	Kondisi sistem jika *use case* berhasil dijalankan.
Failed End Condition	Kondisi sistem jika *use case* gagal dijalankan.
Primary Actors	Aktor utama yang berpartisipasi pada *use case*.
Secondary Actors	Aktor yang berpartisipasi pada *use case* tetapi tidak menjadi yang utama.
Trigger	*Event* yang dipicu oleh aktor yang menyebabkan *use case* dijalankan.
Main Flow	Mendeskripsikan langkah yang penting pada eksekusi normal sebuah *use case*.
Extensions	Mendeskripsikan langkah alternatif suatu langkah di *Main Flow*.

Adapun skenario *use case* dari *use case Create a new Blog Account* dari Gambar 2.5 adalah sebagai berikut:

Tabel 2.2 Contoh Skenario *Use Case* [23]

Use case name	***Create a new Blog Account***
Related Requirements	*Requirement A.1.*
Goal In Context	*A new or existing author requests a new blog account*

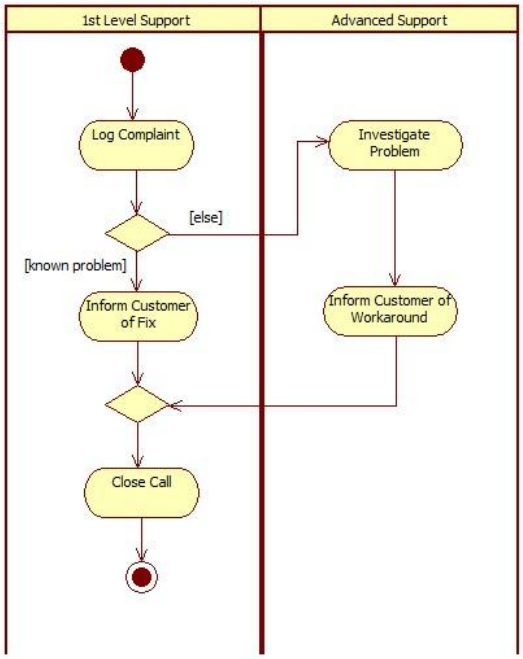
Use case name	***Create a new Blog Account***	
	from the Administrator.	
Preconditions	*The system is limited to recognized authors and so the author needs to have appropriate proof of identity.*	
Successful End Condition	*A new blog account is created for the author.*	
Failed End Condition	*The application for a new blog account is rejected.*	
Primary Actors	*Administrator*	
Secondary Actors	*Author Credentials Database*	
Trigger	*The Administrator asks the CMS to create a new blog account.*	
Main Flow	*Step*	*Action*
	1	*The Administrator asks the system to create a new blog account.*
	2	*The Administrator selects an account type.*
	3	*The Administrator enters the author's details.*
	4	*The author's details are verified using the Author Credentials Database.*
	5	*The new blog account is created.*
	6	*A summary of the new blog account's details are emailed to the author.*
Extensions	*Step*	*Branching Action*
	4.1	*The Author Credentials Database does not verify the author's details.*
	4.2	*The author's new blog account application is rejected.*

2.6.2 *Activity Diagram*

Sebuah *activity diagram* menggambarkan perilaku dinamis dari suatu sistem atau bagian dari sistem melalui aliran kontrol antara tindakan yang sistem lakukan. *Activity diagram* mirip dengan flowchart kecuali bahwa suatu *activity diagram* dapat menunjukkan arus bersamaan.

Komponen utama dari suatu *activity diagram* adalah *action node*, diwakili oleh persegi panjang bulat, yang sesuai dengan tugas yang dilakukan oleh sistem perangkat lunak. Panah dari satu *action node* ke *action node* yang lain menunjukkan aliran kontrol. Artinya, panah antara dua *action node* berarti bahwa setelah aksi pertama selesai, aksi kedua dimulai. Sebuah titik hitam yang solid membentuk simpul awal yang menunjukkan titik awal kegiatan. Sebuah titik hitam yang dikelilingi oleh lingkaran hitam merupakan simpul akhir menunjukkan akhir kegiatan.

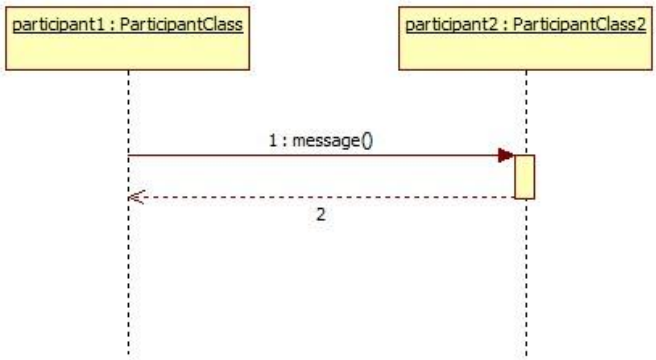
Fork merupakan pemisah kegiatan menjadi dua atau lebih aktivitas bersamaan. Hal ini digambarkan sebagai bar hitam horisontal dengan satu panah menunjuk ke sana dan dua atau lebih anak panah keluar. Setiap panah keluar mewakili aliran kontrol yang dapat dieksekusi bersamaan dengan arus yang sesuai dengan panah keluar lainnya. Kegiatan bersamaan dapat dilakukan pada komputer dengan menggunakan *thread* yang berbeda atau bahkan menggunakan komputer yang berbeda [22].



Gambar 2.7 Contoh Activity Diagram Customer Support [23]

2.6.3 *Sequence Diagram*

Sequence diagram digunakan untuk menunjukkan komunikasi dinamis antara objek selama pelaksanaan tugas. Ini menunjukkan susunan temporal di mana pesan yang dikirim antara objek untuk menyelesaikan tugas tersebut. *Sequence diagram* mungkin digunakan untuk menunjukkan interaksi dalam satu *use case* atau dalam satu skenario dari sistem perangkat lunak.



Gambar 2.8 Contoh Sequence Diagram [23]

Setiap kotak pada baris di bagian atas diagram biasanya sesuai dengan objek, meskipun ada kemungkinan untuk memiliki kotak model yang lain, seperti kelas. Di bawah setiap kotak ada garis putus-putus yang disebut *lifeline* objek. Sumbu vertikal dalam *sequence diagram* sesuai dengan waktu, dengan waktu meningkat saat bergerak ke bawah.

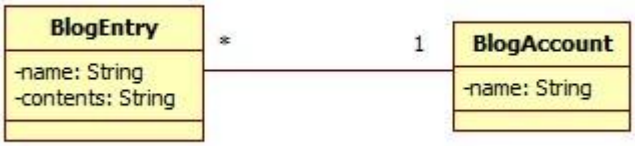
Sequence diagram menunjukkan pemanggilan metode menggunakan panah horisontal dari *caller* kepada *callee*, diberi label dengan nama metode dan secara opsional termasuk parameter, jenis, dan jenis kembali. Ketika sebuah objek mengeksekusi metode (yaitu, ketika memiliki frame aktivasi pada *stack*), dapat menampilkan *bar* putih, disebut *bar* aktivasi, di bawah *lifeline* objek [22].

2.6.4 *Class Diagram*

Untuk memodelkan kelas, termasuk atribut mereka, operasi, dan hubungan dan asosiasi mereka dengan kelas-kelas lain, UML menyediakan *class diagram*. Sebuah *class diagram* memberikan pandangan statis atau struktural dari sebuah

sistem. *Class diagram* tidak menunjukkan sifat dinamis dari komunikasi antara objek dari kelas dalam diagram.

Elemen utama dari *class diagram* adalah kotak, yang merupakan ikon yang digunakan untuk mewakili kelas dan interface. Setiap kotak dibagi menjadi beberapa bagian. Bagian atas berisi nama kelas. Bagian tengah terdapat daftar atribut dari kelas. Sebuah atribut mengacu pada sesuatu yang menjadi obyek dari kelas tersebut. Atribut biasanya diimplementasikan sebagai *field* dari sebuah kelas. Bagian bawah berisi operasi dari kelas. Operasi mengacu pada objek apa dari kelas dapat lakukan, biasanya diimplementasikan sebagai *method* dari sebuah kelas.



Gambar 2.9 Contoh Class Diagram [23]

Setiap atribut dapat memiliki nama, jenis, dan tingkat visibilitas. Jenis dan visibilitas adalah opsional. Jenis ini mengikuti nama dan dipisahkan dari nama dengan titik dua. Visibilitas ditunjukkan dengan sebelumnya -, #, ~, atau +, menunjukkan *private, protected, package,* atau visibilitas *public* [22].

2.6.5 *Component Diagram*

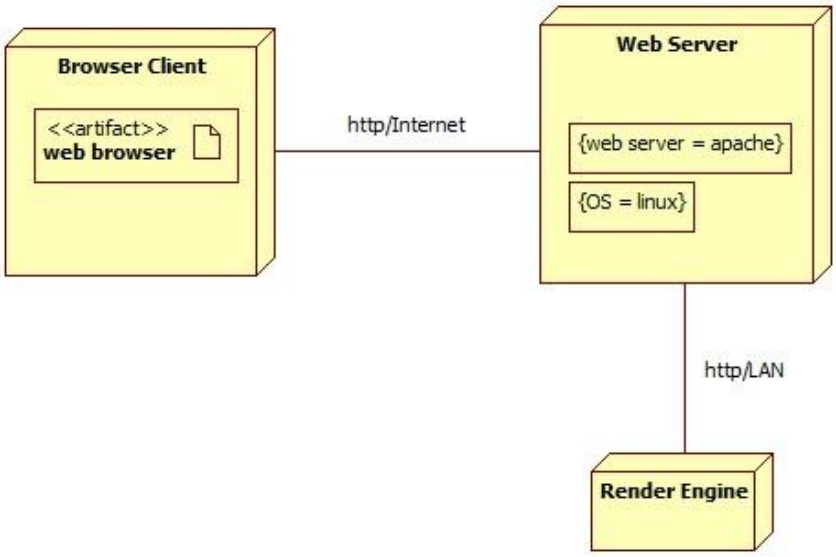
Komponen digunakan untuk mengatur sistem menjadi bagian – bagian yang dapat dikelola, dapat digunakan kembali, dan ditukar dari sebuah sistem. *Component diagram* digunakan untuk memecah suatu sistem menjadi beberapa bagian komponen dan menampilkan hubungan mereka. Secara umum, dapat dikatakan bahwa *component diagram* digunakan untuk menjelaskan kebergantungan antar komponen dalam sebuah sistem [23].



Gambar 2.10 Contoh Component Diagram [23]

2.6.6 *Deployment Diagram*

Sebuah *deployment diagram* berfokus pada struktur sistem perangkat lunak dan berguna untuk menunjukkan distribusi fisik sistem perangkat lunak di antara *hardware* platform dan lingkungan eksekusi. Dalam diagram tersebut, komponen perangkat keras digambarkan dalam kotak berlabel "« device »". Jalur komunikasi antara komponen perangkat keras digambar dengan garis – garis dengan label yang opsional.



Gambar 2.11 Contoh Deployment Diagram [22]

Setiap node dalam sebuah *deployment diagram* juga dapat dijelaskan dengan detail tentang perangkat. Sebuah *artifact* biasanya *file* yang berisi perangkat lunak yang berjalan pada perangkat [22].