BAB II
LANDASAN TEORI

2.1 *Location Based Service* (LBS)

Location Based Service (LBS) atau layanan berbasis lokasi adalah sebuah layanan informasi yang dapat diakses dengan perangkat bergerak melalui jaringan dan mampu menampilkan posisi secara geografis keberadaan perangkat bergerak tersebut. LBS dapat berfungsi sebagai layanan untuk mengidentifikasi lokasi dari seseorang atau suatu objek tertentu, dan juga dapat beraksi aktif terhadap perubahan entitas posisi sehingga mampu mendeteksi letak objek dan memberikan layanan sesuai dengan letak objek yang telah diketahui tersebut. Hal paling penting dari LBS dapat bekerja sesuai yang diinginkan oleh pengembang aplikasi android. Android pun memungkinkan pengembang menentukan metode pencarian lokasi yang dibutuhkan spesifik yang akan dibuat untuk aplikasi tersebut [1].

2.2 Android

Android merupakan 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 [2].

Salah satu elemen kunci dari Android adalah *Dalvik Virtual Machine* (DVM). Android berjalan di dalam *Dalvik Virtual Machine* (DVM) bukan di *Java Virtual Machine* (JVM), sebenarnya banyak persamaannya dengan *Java Virtual Machine* (JVM) seperti Java ME (*Java Mobile Edition*), tetapi android menggunakan *Virtual Machine* sendiri.

Java Delepment Kit (JDK) merupakan bagian terpenting dalam pengembangan aplikasi android, karena android merupakan aplikasi yang dibangun dengan menggunakan bahasa pemrograman java.

Android *Software Development Kit* (SDK) adalah *tools* API (*Application Programming Interface*) yang diperlukan untuk mulai mengembangkan aplikasi pada *platform* android menggunakan bahasa pemgrograman Java. Android

merupakan subset perangkat lunak untuk ponsel yang meliputi sistem operasi, *middleware* dan aplikasi kunci yang di-*release* oleh Google. Saat ini disediakan android SDK sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada platform android menggunakan bahasa pemrograman Java. Sebagai *platform* aplikasi – netral, android memberi anda kesempatan untuk membuat aplikasi bawaan *Handphone*/*Smartphone* [12].

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

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*. *Siklus* hidup *activity* ini sangat penting ketika hendak membuat aplikasi android. Gunanya adalah agar aplikasi yang dibangun berjalan dengan baik. Berikut penjelasan pada masing-masing *state* yang ada di *Android Life Cycle* :

- onCreate(), Method ini dipanggil ketika *activity* pertama kali dibuat.
- onStart(), Method ini dipanggil ketika *activity* tampil ke pengguna.
- onResume(), Method ini dipanggil ketika *activity* yang berjalan pada saat itu dihentikan sementara (paused) dan *activity* sebelumnya dijalankan kembali (resumed). (Hasil dari method onRestart()).
- onPause(), Method ini dipanggil ketika *activity* dihentikan sementara (paused) dan berikutnya ketika berjalan kembali akan berada dalam posisi resume dan memanggil method onResume().
- onStop(), Method ini dipanggil ketika *activity* tidak lagi tampak kepada pengguna.

- onDestroy(), Method ini dipanggil sebelum *activity* dihancurkan(destroy) oleh sistem (baik secara manual maupun untuk kepentingan kelonggaran memori) [4].

Eclipse adalah sebuah *Integrated Development Environment* (IDE) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua *platform* (platform-independent). Eclipse pada saat ini merupakan salah satu IDE favorit dikarenakan gratis dan *open source*, yang berarti setiap orang boleh melihat kode pemrograman perangkat lunak ini. Kelebihan eclipse yang membuatnya populer adalah kemampuannya untuk dapat dikembangkan oleh pengguna dengan komponen yang dinamakan *plugin*.

2.3 *Google Maps*

Google Maps adalah layanan gratis yang diberikan oleh google dan sangat populer. Google maps adalah peta dunia yang dapat digunakan untuk melihat suatu daerah menggunakan suatu *browser*. Dapat juga menambahkan *fitur Google Maps* dalam *web* yang telah dibuat pada *blog* yang berbayar ataupun yang gratis sekalipun dengan *Google Maps API*. *Google maps API* adalah suatu *library* yang berbentuk *javascript*. Dengan menggunakan *Google Maps API* dapat menghemat waktu dan biaya sehingga hanya berfokus pada data-data yang akan ditampilkan [3].

Pada *google maps api* terdapat 4 jenis pilihan model peta yang disediakan oleh Google, diantaranya adalah sebagai berikut.:

- *Roadmap*, untuk menampilkan peta biasa dalam dua dimensi.
- *Satellite*, untuk menampilkan foto satelit.
- *Terrain*, untuk menunjukkan relief fisik permukaan bumi dan menunjukkan seberapa tingginya suatu lokasi, contohnya akan menunjukkan gunung dan sungai.
- Hybrid, akan menunjukkan foto satelit yang diatasnya tergambar pula apa yang tampil pada *roadmap* (jalan dan nama kota).

2.4 *Global Positioning System (GPS)*

GPS (*Global Positioning System*) adalah sistem satelit navigasi dan penentuan posisi yang dimiliki dan dikelola oleh amerika serikat. Sistem ini di desain untuk memberikan posisi dan kecepatan tiga dimensi serta informasi mengenai waktu, secara *continue* di seluruh dunia tanpa bergantung waktu dan cuaca, bagi banyak orang secara simultan GPS dapat memberikan informasi posisi dengan ketelitian bervariasi dari beberapa milimeter sampai dengan puluhan meter. Ketelitian dari GPS dapat mencapai beberapa milimeter untuk ketelitian posisinya, beberapa cm/s untuk ketelitian kecepatannya dan beberapa nanodetik untuk ketelitian waktunya. Ketelitian posisi yang diperoleh akan tergantung pada beberapa faktor yaitu metode penentuan posisi, geometri satelit, tingkat ketelitian data dan metode pengolahan datanya [3].

2.5 Konsep *Graph*

Teori *graph* adalah cabang ilmu yang mempelajari sifat-sifat *graph*. Secara informal, suatu *graph* adalah himpunan benda-benda yang disebut *vertex* (*node*) yang terhubung oleh *edge-edge* (*line*). Biasanya *graph* digambarkan sebagai kumpulan titik (melambangkan *vertex*) yang dihubungkan oleh garis-garis (melambangkan *edge*). Definisi yang lebih formal adalah suatu *graph* G yang dapat dinyatakan sebagai G = <V,E>. *Graph* G terdiri atas himpunan V yang yang berisikan *vertex/node* pada *graph* tersebut dan himpunan dari E yang berisi *edge* pada *graph* tersebut. Himpunan E dinyatakan sebagai pasangan dari *vertex* [5].

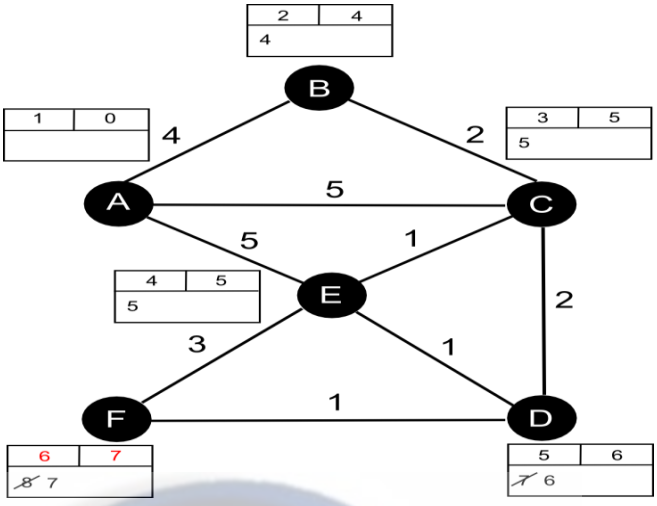
Jenis-jenis *graph* yaitu *graph* berarah dan berbobot, *graph* tidak berarah dan berbobot, *graph* berarah dan tidak berbobot, dan *graph* tidak berarah dan tidak berbobot. *Graph* yang digunakan dalam metode perhitungan ini yaitu menggunakan *graph* berarah dan berbobot. Tiap *edge* mempunyai anak panah dan mempunyai bobot. bobot tersebut adalah bilangan positif jadi tidak dapat dilalui oleh vertex/*node* negatif. Namun jika terjadi demikian, maka penyelesaian yang diberikan adalah *infiniti* (Tak Hingga) [5].

2.6 Metode *Algoritma Djikstra*

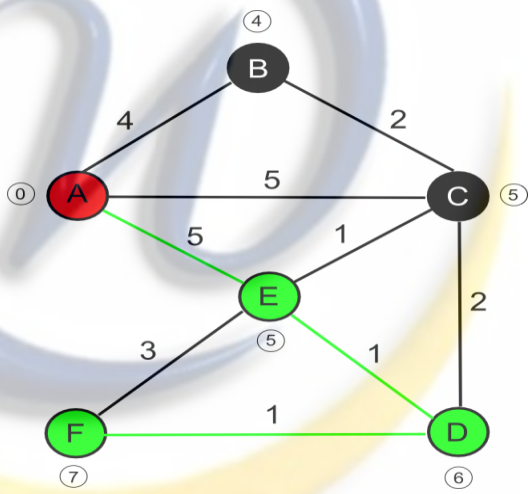
Dalam pembuatan aplikasi ini dibutuhkan suatu teknik perhitungan untuk mendapatkan nilai jarak lokasi tempat terhadap lokasi pengguna, dimana teknik perhitungan tersebut menggunakan rumus *Algoritma Djikstra*. *Algoritma Djikstra* adalah suatu algoritma yang digunakan untuk mencari lintasan terpendek untuk mencapai titik tujuan dari titik sumber pada sebuah *graph*. Pada praktek nya algoritma ini tidak hanya mencari lintasan pendek dari sumber ke tujuan, juga dapat mencari lintasan terpendek dari sumber ke semua titik pada *graph*. Dalam proses menemukan semua jalan terpendek untuk semua tujuan, akan terbentuk pohon lintasan terpendek (*spanning tree*) sebagai hasil akhir dari *algoritma djikstra* yang menjadi *root* adalah sumber sedangkan yang menjadi *leaf* adalah titik tujuan [6].

Langkah – langkah perhitungan algoritma djikstra adalah :

1. Inisialisasi beberapa *vertex/node*, *vertex/node* yang bisa dijangkau secara langsung dan juga jarak antara mereka.
2. Inisialisasi *vertex/node* daerah awal.
3. Inisialisasi *vertex/node* daerah tujuan.
4. Beri tabel sementara untuk jarak antar *vertex/node*.
5. Menentukan jarak minimum dari tabel sementara
6. Beri tabel permanen untuk vertex/node dengan jarak minimum tersebut serta beri urutan tabel.
7. Menghapus daftar tabel sementara
8. Mencari *vertex/node* minimum berikutnya dengan membandingkan jarak menuju *node* atau melalui titik yang telah memiliki tabel permanen.
9. Jika ada maka ulangi langkah ke-6 begitu seterus nya sampai diktahui *vertex/node* terakhir.



Gambar 2.1 Perhitungan Algoritma Djikstra



Gambar 2.2 Rute yang dilewati

2.7 *Object Oriented Analysis and Design*

Analisis dengan disain 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. Tools yang dapat digunakan pada pendekatan analisis pengembangan sistem secara objek dapat menggunakan UML [8].

2.7.1 *Unified Modeling Language* (UML)

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML tidak berdasarkan pada bahasa pemrograman tertentu. UML yang berorientasikan objek mempunyai beberapa notasi standar.

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 :

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 [9].

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 use *case*nya.

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai *respon* dari sebuah *event* untuk menghasilkan *output* tertentu. *Message* digambarkan sebuah garis berpanah dari satu objek ke objek lainnya. Berikutnya *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki ciri-ciri khusus, standar UML mendefinisikan icon khusus untuk objek *boundary, controller* dan *persistent entity* [9].

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) suatu 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 c*ontainment*, pewarisan, asosiasi, dan lain-lain [9].

Class memiliki tiga area pokok yaitu Nama, Atribut dan Metoda. Atribut dan metoda dapat memiliki salah satu sifat berikut :

a. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
b. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
c. *Public*, dapat dipanggil oleh siapa saja

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.

2.8 *SQLite*

SQLite adalah *database* yang berukuran kecil, berdiri sendiri bukan *database client server*, tanpa konfigurasi namun mempunyai fitur penuh perintah SQL. SQLite adalah produk *public* domain yang tidak mempunyai lisensi dan boleh mengambil *binary* atau *sourcecode*nya secara *free*. SQLite bisa digunakan di semua *platform* seperti windows, linux, android atau bahkan iPhone. *Tools* yang populer untuk *commandline* di SQLite adalah SQLmanager. SQLmanager adalah *plugin* firefox. Contoh produk yang menggunaan SQLite adalah PHP, firefox, chrome, iPhone, dan android. Di browser firefox SQLite digunakan untuk menyimpan *konfigurasi*, *bookmark* dan *history website* sedangkan di *smartphone* android SQLite digunakan untuk menyimpan *contact*. Untuk android, SQLite

dijadikan satu di dalam android *runtime* sehingga setiap aplikasi android dapat membuat basis data SQLite [7].

Terdapat beberapa alasan mengapa SQLite cocok untuk pengembangan android yaitu :

- *Database* dengan konfigurasi 0. Artinya tidak ada konfigurasi *database* untuk *developer*.
- Tidak memiliki *server*.
- Tidak ada proses *database* SQLite yang berjalan.
- Pada dasarnya satu *set libraries* menyediakan fungsionalitas *database*.
- *Single-file database*. Ini membuat keamanan *database* secara langsung.
- *Open source*. Hal ini membuat *developer* mudah dalam pengembangan aplikasi [7].

2.9 Pengujian *Black box*

Untuk aplikasi yang akan dibuat menggunakan teknik pengujian *black box*. Metode pengujian *black box* memfokuskan pada keperluan fungsional dari *software* [8]. Ujicoba *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 [10].