BAB 2
TINJAUAN PUSTAKA

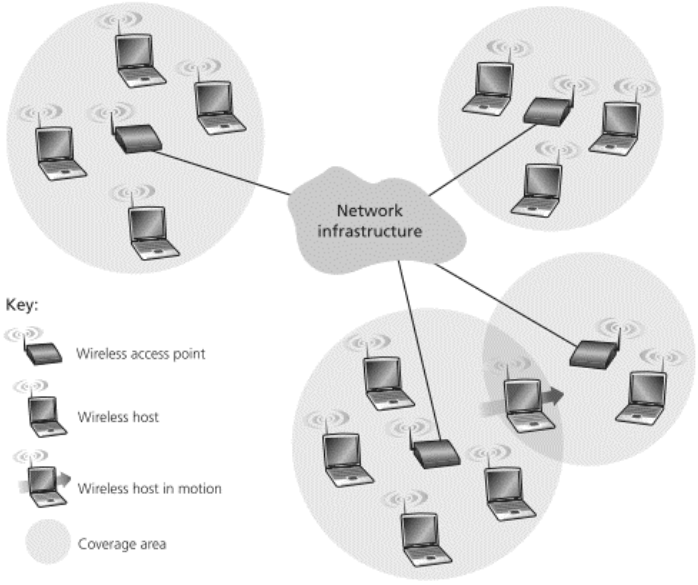
2.1 Pengenalan Jaringan Nirkabel dan Wi-Fi

Jaringan nirkabel *(Wireless Network)* adalah salah satu cara untuk melakukan komunikasi data dalam jaringan komputer tanpa melalui kabel. Contoh dari jaringan nirkabel adalah jaringan komunikasi pada telepon seluler dan Wi-Fi. Terdapat elemen-elemen penting dalam membentuk jaringan nirkabel, yaitu :

1. *Wireless Host*

   Dalam jaringan kabel, *host* adalah perangkat akhir yang menjalankan aplikasi. Sedangkan dalam jaringan nirkabel, *host* bisa berupa *smartphone, tablet,* dan laptop.
2. *Wireless Links*

   Adalah sebuah host yang terkoneksi ke *Base Station* yang kemudian akan dijelaskan selanjutnya. Atau *host* yang terkoneksi ke *host* lainnya melalui jaringan komunikasi data nirkabel.



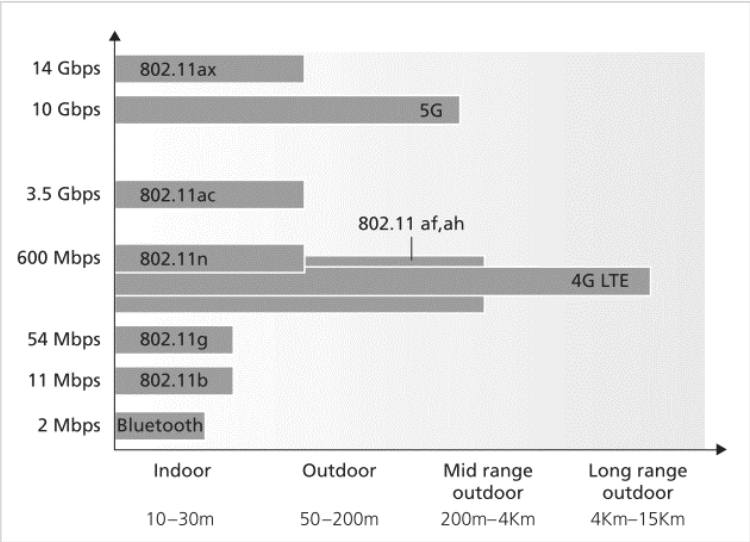
Gambar 2.1 Elemen-Elemen Jaringan Nirkabel

Gambar 2.1 memperlihatkan elemen-elemen yang terlibat dalam membentuk infrastruktur jaringan nirkabel. *Network Infrastructure* adalah suatu jaringan

besar internet, digunakan oleh *Base Station* agar bisa terhubung ke dalam jaringan yang lebih besar, bukan sekedar jaringan lokal. *Wireless Access Point* bertugas sebagai *Base Station*, dan kemudian terlihat dengan jelas pada gambar tersebut bahwa *Wireless Host* akan terkoneksi dengan *Base Station* proses koneksi ini kemudian dinamakan *Wireless Links.*

3. *Base Station*

   Merupakan elemen utama dalam infrastruktur jaringan nirkabel. *Base Station* bertanggung jawab dalam mengatur pengiriman dan penerimaan data ke dan dari *Wireless Host* yang terkoneksi ke *Base Station.* Karena elemen ini yang memegang tanggung jawab tersebut, maka elemen ini juga berperan sebagai pemancar sinyal yang akan diterima oleh *Wireless Host.*



Gambar 2.2 Standar Kecepatan dan Jangkauan Jaringan Nirkabel

Gambar 2.2 memperlihatkan standar-standar teknologi yang ada pada jaringan nirkabel, di mana setiap standar memiliki kemampuan berdasarkan kecepatan dan jangkauan sinyalnya. Pada gambar tersebut terdapat standar *802.11 Wireless Local Area Network (LAN)* yang kemudian dikenal dengan sebutan *Wireless-Fidelity (Wi-Fi).* Standar ini kemudian digunakan dalam setiap perangkat yang mendukung Wi-Fi, salah satunya *Base Station* atau *Wireless Access Point* itu sendiri, jika dilihat pada gambar Gambar 2.1. Contoh

perangkat *Base Station* adalah perangkat yang mendukung standar 802.11 seperti *Wireless Router* dan Telepon Seluler atau *Smartphone.*

4. *Network Infrastructure*

   Network Infrastructure pada Gambar 2.1 merupakan jaringan komputer yang lebih besar yang memungkinkan Wi-Fi yang sebelumnya hanya merupakan infrastruktur jaringan lokal dapat terhubung ke jaringan lebih besar seperti internet [7].

Dari penjelasan definisi serta elemen-elemen jaringan nirkabel di atas dapat diketahui bahwa Wi-Fi adalah salah satu bagian dari komunikasi jaringan nirkabel yang menggunakan standar 802.11. Wi-Fi merupakan elemen kunci dalam penelitian untuk membangun aplikasi berbagi Wi-Fi gratis yang sehingga landasan ini sangat penting sebagai acuan dalam penelitian.

2.2 Jaringan Iklan Daring

Jaringan iklan daring merupakan sebuah konsep layanan di mana penyedia layanan bertujuan untuk dapat menghubungkan penayang iklan yang bersedia menjual *impression* dari penayangan iklan seperti *click,* atau *views* ke pengiklan yang menginginkankan seseorang untuk menayangkan iklan miliknya [8]. Dari definisi tersebut dapat diketahui bahwa konsep ini dijalankankan dengan cara pengiklan membayarkan sejumlah dana ke penyedia layanan jaringan iklan daring untuk dapat dicarikan seseorang yang mau menampilkan iklan miliknya di media online seperti *website* atau aplikasi smartphone, lalu penayang iklan akan mulai menayangkan iklan dari media yang dimilikinya. Dilihat dari konsep ini setiap pihak yang terlibat dapat saling diuntungkan. Salah satu penyedia layanan jaringan iklan daring ini adalah Google yang menamai layanannya dengan Google Adwords untuk pengiklan dan Google Adsense untuk penayang iklan *website* dan video streaming, dan Google AdMob untuk penayang iklan smartphone.

Dengan adanya landasan teori mengenai bagaimana konsep jaringan penayangan iklan tersebut, maka hal ini dapat sangat membantu dalam penelitian untuk membangun konsep bagaimana berjalannya aplikasi berbagi Wi-Fi gratis

ini nanti yang kemudian peneliti di sini akan memilih Google AdMob dengan opsi media smartphone sebagai media penayangan iklan nanti di dalam aplikasinya.

2.2.1 Format Iklan Daring di *Smartphone*

Format iklan merupakan penentu bagaimana bentuk iklan yang nanti akan ditampilkan di dalam aplikasi *smartphone.* Dalam *smartphone* sendiri penyedia jaringan iklan daring menyediakan beberapa format tampilan iklan yang dapat disesuaikan dengan kebutuhan yang ada di dalam aplikasi. Berikut adalah format iklan yang ada tersebut :

1. *Banner*

   Format iklan ini merupakan format dasar yang umum digunakan, iklan akan ditampilkan di bagian atas dan bawah aplikasi di layar perangkat.
2. *Interstisial*

   Format iklan ini menampilkan iklan satu layar penuh, dapat berupa video, link, atau hanya gambar. Biasanya akan tampil ketika pengguna berpindah halaman dalam suatu aplikasi.
3. *Reward*

   Format iklan ini memungkinkan pengguna dapat berinteraksi langsung dengan iklan, bisa berupa *survey* atau *download* aplikasi. Dari hasil interaksi tersebut pengguna dapat diberikan *reward.* Contoh *reward* ini bisa berupa apapun, misalkan dalam aplikasi yang dibangun saat ini, seseorang akan mendapatkan *reward* tambahan jangka waktu muncul iklan.
4. *Native*

   Iklan dengan format ini akan tampil secara dinamis dengan cara disesuaikan melalui tampilan aplikasi tempat iklan tersebut muncul.

Dari ke empat format iklan tersebut, guna mendukung pengembangan aplikasi ini maka penulis memilih menggunakan format *Interstisial* dan *Reward* agar pengguna bisa melihat iklan satu layar penuh.

2.2.2 Model Pendapatan Iklan Daring

Model pendapatan yang diterapkan dalam iklan daring adalah cara perhitungan pendapatan penayang iklan dari hasil penayangan yang dilakukannya. Di dalam jaringan iklan daring, terdapat beberapa model pendapatan yang telah diterapkan, terutama pada jaringan iklan daring yang disediakan oleh Google AdMob. Berikut ini adalah model-model pendapatan yang diterapkan di dalamnya:

1. ***Cost-per-Mille (CPM)***

   Mille dalam bahasa Latin, Perancis, dan Italia berarti ribu. Dengan menggunakan model ini, penayang iklan akan mendapatkan pendapatan setiap kali iklannya telah mencapai seribu *impression* atau pengaruh atau penayangan. Untuk mendapatkan nilai estimasi pendapatan per seribu kali tayang, dapat menggunakan rumus berikut :

   $$CPM = \frac{Estimasi\ Pendapatan}{Jumlah\ Tayangan} \times 1000$$

   Contoh, jika estimasi pendapatan yang diperkirakan oleh pihak penyedia layanan iklan adalah $0.15 dari 25 tayangan, maka dapat diperkirakan dengan rumus tersebut jumlah pendapatan untuk setiap seribu kali tayang sebesar $6.

2. ***Cost-per-click (CPC)***

   Model ini menerapkan perhitungan pendapatan bagi penayang iklan untuk setiap kali iklan diklik. Jumlah klik akan sangat menentukan penghasilan yang didapatkan. Untuk menghitung perkiraan pendapatan per satu kali klik iklan yang ditayangkan, dapat dilakukan dengan cara mencari persentase sebeerapa sering iklan yang ditampilkan diklik atau disebut juga *Click-through Rate (CTR)* terlebih dahulu.

   $$CTR\ (\%) = 0{,}01 \times \frac{Jumlah\ Klik}{Jumlah\ Tayangan}$$

   Setelah persentase diketahui, baru dapat dilakukan perhitungan untuk mendapatkan estimasi CPC dengan menggunakan rumus berikut :

   $$CPC = (\frac{CPM}{1000})/(\frac{CTR}{100}) = 0{,}1 \times \frac{CPM}{CTR}$$

Berdasarkan dua model tersebut dapat disimpulkan bahwa keduanya saling berkaitan erat dan keduanya digunakan pihak penyedia iklan terutama oleh Google AdMob. Namun, perlu diketahui bahwa rumus-rumus di atas hanya untuk membantu mendapatkan nilai estimasi, untuk nilai pastinya tergantung algoritma yang diterapkan oleh penyedia iklan. Estimasi perhitungan pendapatan di atas akan sangat berguna dalam penelitian, terutama dalam menentukan pedapatan yang akan diterima oleh pemilik Wi-Fi yang bersedia membagikan koneksinya secara gratis.

Pada tahun 2021, salah satu perusahaan penyedia layanan jaringan iklan daring [2] telah melakukan riset untuk mencari performa indeks pengiklanan pada setiap layanan penyedia iklan, seperti pada Google AdMob berdasarkan berdasarkan negara, format iklan, model pendapatan, dan sistem informasi perangkat smartphone.

Gambar 2.3 Estimasi CPM 2021 di Indonesia

Pada Gambar 2.3 menunjukkan bawa dalam laporan hasil riset tersebut diketahui format iklan Interstitial di negara Indonesia memiliki nilai CPM paling besar mencapai $1,22. Dan berdasarkan data tersebut dapat dilakukan estimasi perhitungan pendapatan yang nantinya dapat diimplementasikan dalam penelitian

ini. Untuk melakukan estimasi pendapatan untuk setiap penyedia Wi-Fi dapat menggunakan rumus berikut ini:

1. Menghitung jumlah penayangan iklan dalam kurun waktu tertentu (N)

   $$N = \frac{T}{M} \times P \times W$$

   Keterangan:

   N = Jumlah penayangan iklan

   T = Lama penggunaan jaringan (menit)

   M = Interval iklan muncul (menit)

   P = Jumlah pengguna jaringan

   W = Lama penayangan (hari)

2. Menghitung jumlah pendapatan (S) berdasarkan jumlah penayangan iklan

   $$S = \left(\frac{N}{1000}\right) \times CPM$$

   Keterangan:

   S = Jumlah pendapatan

   N = Jumlah penayangan iklan

   CPM = *Cost per mille* (estimasi pendapatan per seribu kali tayang)

Pembahasan mengenai model pendapatan pada iklan daring ini sangat penting untuk dicantumkan dalam landasan teori. Karena tujuan dari penelitian yang sebisa mungkin dapat mendorong penyedia Wi-Fi untuk membagikan jaringannya secara gratis. Sehingga konsep perhitungan di atas sangat penting untuk diketahui.

2.3 Android

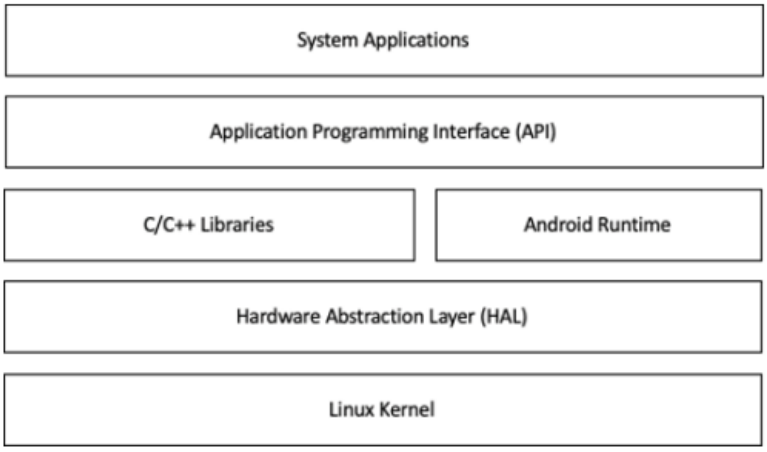
Android adalah sistem operasi multi perangkat yang menggunakan kernel atau inti dari hasil modifikasi sistem operasi Linux. Awalnya pada tahun 2003, Android dikembangkan oleh Andy Rubin dan koleganya sebagai sistem operasi kamera *Digital Single-Lens Reflex (DSLR)* dan diberi nama “FotoFarm”, dan saat itu juga telah memiliki situs dengan domain android.com. Di bulan November 2004, Andy dan kolega melakukan pertemuan dan berdiskusi bahwa minat pada sistem operasi kamera tidak terlalu baik. Karena hal tersebut, mereka menentukan

fokus sistem operasi selanjutnya adalah telepon [9]. Sampai pada tahun 2005, karena melihat peluang pada Android kemudian google mulai mendanai pengembangan sistem operasi tersebut sampai mereka memutuskan untuk mengakuisisinya.

Setelah berdiri di bawah naungan Google, Android selama beberapa tahun terlihat tidak memiliki perkembangan, namun akhirnya pada tahun 2008 diluncurkanlah smartphone pertama dengan sistem operasi Android bernama HTC Dream. Karena kurangnya peminat pada smartphone tersebut, sehingga Android belum terlalu dikenal dan pada tahun-tahun tersebut juga masih kalah tenar dengan sistem operasi BlackBerry. Karena kegigihan dari tim pengembang, diluncurkanlah terus menerus smartphone berbasis Android seperti Nexus One, Samsung, dan lain-lain. Hingga saat ini sistem operasi Android berkembang pesat dan dapat digunakan diberbagai jenis perangkat mulai dari smartphone, smartwatch, televisi, dan lain sebagainya. Hal ini menandakan bahwa sistem operasi Android cukup fleksibel dan mendukung kebebasan dalam pengembangannya.

2.3.1 Android *Stack*

Sistem operasi android di dalamnya terdiri dari beberapa *stack* atau lapisan. Setiap lapisan memiliki fungsi masing-masing dan saling melengkapi sehingga membentuk suatu sistem informasi yang solid. Penting untuk diketahui lapisan-lapisan yang ada di dalam Android, hal ini akan berguna dalam pengembangan aplikasi dan sistem di Android. Karena dengan mengentahui layer-layer tersebut pengembang akan tahu bagian-bagian apa saja yang akan disentuh dan dimanfaatkan selama pengembangan.



Gambar 2.4 Lapisan di dalam sistem informasi Android

Gambar 2.4 memperlihatkan lapisan-lapisan yang ada di dalam sistem operasi Android. Pada lapisan paling bawah dapat dilihat bahwa Android dibangun di atas Linux *Kernel* atau inti Linux [10]. Lebih lengkapnya, berikut ini adalah penjelasan terkait masing-masing lapisan di atas :

1. Linux *Kernel*

   Bagian inti dari Android adalah Linux Kernel, berperan mengelola segala jenis hardware yang berjalan di dalam perangkat.
2. *Hardware Abstraction Layer (HAL)*

   Merupakan sebuah sistem penterjemah universal. Android Runtime dapat berjalan di setiap jenis perangkat, tapi Android Runtime tidak dikembangkan untuk segala jenis perangkat. Untuk bisa berlajan, Android Runtime memerlukan lapisan ini.
3. C/C++ *Library* dan Android *Runtime*

   Pada lapisan tengah ini terdiri dari library yang berperan memberikan dukungan library pada lapisan ke-empat. Contoh dari library ini seperti SQLite, Webkit, dan OpenGL. Sedangkan Android Runtime adalah lapisan yang bertanggung jawab melakukan kompilasi aplikasi di awal, sehingga setiap kali aplikasi dijalankan, Android tidak perlu melakukan kompilasi secara berkala.
4. *Application Programming Interface (API)*

Aplikasi android yang dijalankan dan dikembangkan akan membuat request ke dalam lapisan ini. Di dalam lapisan ini terdapat API yang memungkinkan program mengakses berbagai fitur yang ada di dalam Android, seperti GPS, Wi-Fi, Telepon, Internet, Activity dan Fragment Manager, dan lain sebagainya.

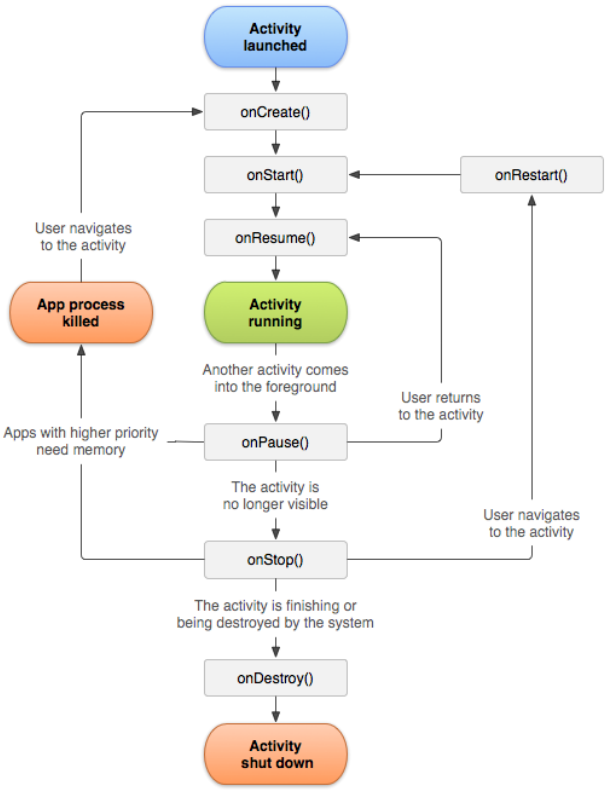
5. *System Application*

   Lapisan ini adalah tempat dari segala aplikasi yang ada di Android. Pengguna dan pengembang aplikasi akan berinteraksi langsung dengan lapisan ini, pengguna berperan sebagai orang yang menjalankan aplikasi, sedangkan pengembang berperan sebagai penulis kode dan membuatnya menjadi sebuah aplikasi yang siap dijalankan.

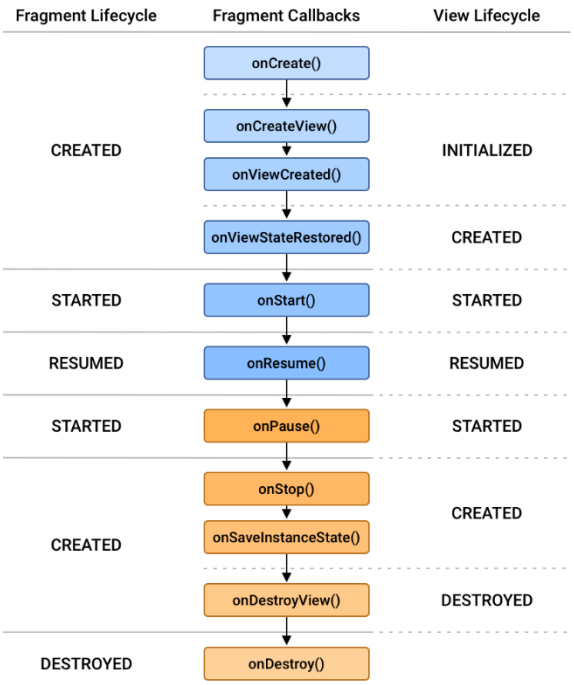
2.3.2 Android Life -Cycle

Dalam pengembangan aplikasi Android, tidak terlepas dengan namanya Activity dan Fragment. Activity adalah sebuah halaman di dalam aplikasi yang dapat dilihat oleh penggunanya, sedangkan Fragment diciptakan khusus untuk mengatasi ukuran layar perangkat yang berbeda-beda. Alih-alih aplikasi harus membuat ulang suatu Activity untuk setiap ukuran layar, maka akan lebih baik dengan menggunakan Fragment.

Activity dan Fragment dijalankan melalui suatu proses yang panjang sehingga membutuhkan sumber daya perangkat yang banyak. Karena proses pembuatannya yang begitu membutuhkan banyak sumber daya, akan sangat disayangkan jika hanya melakukan suatu tindakan sampai Activity dan Fragment dijalankan. Maka dari itu, diciptakanlah Life-Cycle dalam proses pembuatan Activity dan Fragment [10]. Dengan melalui Life-Cycle, pengembang dapat ikut serta dalam tahapan pembuatan Activity dan Fragment di belakang layar.



Gambar 2.5 Ilustrasi Life-Cycle dalam Activity



Gambar 2.6 Ilustrasi Life-Cycle dalam Fragment

Pada Gambar 2.5 dan Gambar 2.6 memperlihatkan ilustrasi *Life-Cycle* yang dilalui selama proses pembuatan *Activity* dan Fragment di dalam aplikasi. Setiap tahapan di dalamnya memiliki suatu *Callbacks* yang dengan melalui itu pengembang dapat ikut menyisipkan kode program ke dalamnya selama proses *Life-Cycle* berjalan.

2.3.3 Location Service API

Android memberikan kebebasan kepada pengambang aplikasi untuk dapat mendapatkan informasi lokasi perangkat. Untuk mendapatkan informasi tersebut, android mengembangkan Location Service API yang telah menemani android sejak awal mula rilis. Informasi yang diberikan melalui API ini berupa informasi letak geografis perangkat dan diperbarui secara periodik. Cara penentuan letak ini berasal dari tiga sumber, yaitu GPS, *Cell Tower Triangulation,* dan Wi-Fi. Melalui ketiga sumber tersebut sistem operasi android dapat menemukan letak perangkat berada saat itu.

Terdapat dua metode yang dapat digunakan oleh pengembang aplikasi dalam mendapatkan informasi lokasi perangkat, yaitu :

1. *Precise*

   Metode ini dapat memberikan informasi lokasi secara presisi dengan memadukan ketiga sumber lokasi.
2. *Approximate*

   Metode ini dapat memberikan informasi perkiraan lokasi perangkat. Umumnya metode ini hanya mengandalkan sumber berdasarkan *Cell Tower Triangular,* atau konsep perhitungan lokasi berdasarkan jarak tiga tower jaringan internet.

Tidak semena-mena memberikan akses terhadap lokasi, dan guna memberikan keamanan privasi bagi pengguna perangkat, Android harus meminta persetujuan pengguna jika ingin mengakses informasi lokasi perangkat. Proses permintaan persetujuan ini dapat dilakukan dengan cara mendaftarkan Location

Services API ketika aplikasi dalam pengembangan, kemudian secara otomatis aplikasi akan menampilkan informasi persetujuan kepada pengguna pada saat menggunakan aplikasi.

2.3.4 Wi-Fi Manager API

Sekarang dalam semua perangkat telepon pintar telah memiliki WI-Fi yang terpasang di dalamnya dan sistem Android memberikan keleluasaan dalam hal melakukan pengelolaan terhadap Wi-Fi yang tertanam di dalam perangkat. Untuk dapat melakukan hal tersebut, android menyediakan kelas bernama Wi-FiManager. Merupakan sebuah kelas yang menyediakan API utama untuk mengelola segala aspek dalam hal konektivitas Wi-Fi.

Kemampuan yang diberikan oleh Android melalui API ini adalah:

1. Mendapatkan daftar seluruh aktifitas koneksi Wi-Fi
2. Melihat dan mengubah detail spesifik suatu koneksi Wi-Fi
3. Melihat jaringan Wi-Fi yang sedang digunakan termasuk detail di dalamnya.
4. Mendapatkan daftar akses poin yang terjaring oleh Wi-Fi perangkat.

2.3.5 Android Studio

Tools atau alat selama pembuatan aplikasi Android sangatlah diperlukan agar proses pembuatannya tidak terlalu memerlukan banyak usaha, terutama dalam hal menghafal kode dan dokumentasi. Untuk dapat melakukan hal tersebut, pengembang harus menggunakan alat yang namanya *Integrated Development Environment (IDE).*

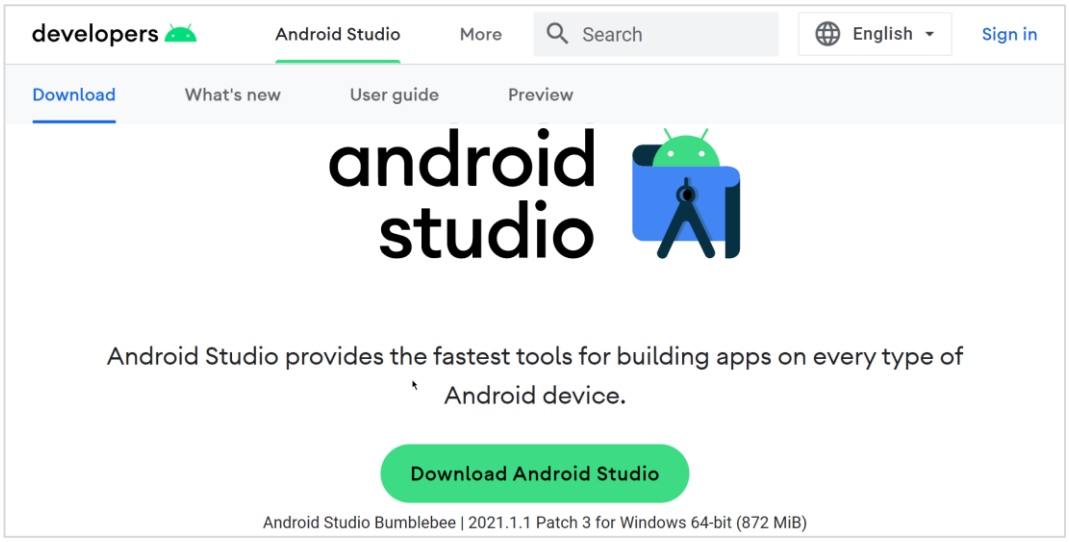
Untuk dapat mengembangkan aplikasi, IDE yang pengembang pilih harus benar dan sesuai dengan kebutuhan pengembanan aplikasi. Maka dari itu, Google memberikan solusi IDE yang cocok untuk digunakan dalam pengembangan aplikasi Android. Melalui IDE ini pengembang tanpa harus memikirkan pengaturan yang kompleks untuk dapat memulai pengembangan. Alat ini bernama Android Studio [10].

Android Studio merupakan IDE yang telah lama ikut berkembang bersama Android. Kabar baiknya, software ini dapat mendukung banyak sistem operasi

untuk dapat menjalankannya. Berikut ini adalah daftar sistem operasi yang didukung oleh Android Studio.

1. Windows 7, 8, atau 10 (32-bit atau 64-bit)
2. Mac OS
3. Linux
4. Chrome OS

Untuk dapat menggunakan alat ini, pengembang dapat mengunduhnya langsung melalui laman Android Studio di https://developer.android.com/studio.



Gambar 2.7 Laman Android Studio

Gambar 2.7 memperlihatkan versi terakhir Android Studio pada saat penelitian ini dilakukan adalah Versi 2021.1.1 Patch 3 bulan April 2022, dengan ukuran sebesar 872 *Megabytes* untuk sistem operasi Windows.

Pengetahuan mengenai Android dan segala aspek di dalamnya yang di mana sebagiannya telah dijelaskan di atas penting untuk diketahui, karena dalam penelitian ini akan menerapkan Android sebagai basis utama pembuatan aplikasi dan dapat mendukung segala kebutuhan selama proses pengembangan berlangsung.

2.4 Java

Java merupakan bahasa pemrograman yang dapat digunakan untuk berbagai tujuan umum. Memiliki kemampuan untuk dapat diimplementasikan dalam berbagai macam paradigma pemrograman. Seperti pemrograman berbasis objek, prosedural, dan fungsional. Java umum dikenal sebagai bahasa pemrograman berbasis objek. Maksud dari itu adalah bahwa bahasa pemrograman ini mendukung untuk digunakan dalam paradigma berbasis objek. Sementara itu, Java sendiri bukanlah bahasa pemrograman berbasis objek, karena java masih bisa digunakan di luar paradigma berbasis objek.

Versi awal Java dirilis oleh Sun Microsystem di tahun 1995. Kemudian di tahun 2010 berpindah ke Oracle. Sementara itu, pengembangan bahasa pemrograman ini telah dimulai semenjak tahun 1991. Dan awalnya, diberinama Oak. Setelah beberapa waktu, para pengembang memutuskan untuk mengganti namanya yang kemudian sampai sekarang dikenal dengan nama Java [11].

Segera setelah rilis, Java menjadi bahasa pemrograman yang populer. Poin yang menjadikannya populer saat itu adalah karena fitur *"Write once, run anywhere"* (WORA), yaitu dengan fitur ini *programmer* cukup menulis program satu kali saja dan dapat menjalankan programnya di manapun meskipun sistem operasinya berbeda. Contoh, seorang *programmer* membuat aplikasi dengan bahasa pemrograman Java di sistem operasi Linux dan dapat dijalankan di Windows dan macOS. Fitur ini dapat dicapai karena Java melakukan kompilasi kode program ke dalam bentuk *bytecode.* Format bytcode ini independen dari segala jenis platform. Untuk menjalankan bentuk *bytecode,* Java menggunakan *Java Virtual Machine (JVM).* Tugasnya untuk mentransofrmasi *bytecode* ke dalam format yang bisa dieksekusi berdasarkan platform tempat kode itu dijalankan.

Sebagai pengembang yang menggunakan Java sebagai pilihan bahasa pemrogramannya, maka mereka harus mengetahui istilah-istilah yang menjadi aturan penting dalam pengembangan aplikasi menggunakan bahasa pemrograman Java [12]. Istilah dan aturan tersebut adalah:

1. *Java Virtual Machine (JVM)*

   Ketika kode Java dikompilasi, akan menghasilkan file dengan ekstensi .class. Di dalam file tersebut berisi *Java Bytecode,* file ini akan dibaca oleh JVM. JVM bertanggung jawab untuk melakukan pembacaan, pemeriksaan, dan menjalankan kode tersebut. JVM merupakan mesin virtual yang independen dari platform, karena JVM dapat membaca Bytecode dan mengkonversinya menjadi bahasa mesin yang sesuai dengan mesin tempat kode tersebut dijalankan.
2. *Java Runtime Environment (JRE)*

   JRE berisi JVM, library, dan file pendukung lainnya. Untuk dapat menjalankan program Java, maka suatu mesin atau komputer harus terpasang JRE terlebih dahulu.
3. *Java Development Kit (JDK)*

   Berperan menyediakan alat yang dibutuhkan untuk mengembangkan aplikasi Java. Di dalam JDK jgua berisi JRE.
4. *Bytecode*

   Merupakan hasil kompilasi dari bahasa pemrograman Java. Umumnya memiliki ekstensi .class. Karena Bytecode akan dibaca oleh JVM, maka dapat disimpulkan bahwa Bytecode adalah bahasa dari JVM.
5. Platform

   Dalam Java, Platform diartikan sebagai tempat dimana program akan dijalankan. Itu dapat berupa mesin, sistem operasi, dan sebagainya.

Karena memiliki fitur WORA, bahkan dalam platform Android pembuatan aplikasinya dapat ditulis menggunakan bahasa pemrograman Java. Dengan adanya platform Android yang mendukung penggunaan Java, menjadikan bahasa pemrograman ini semakin populer di kalangan pengembang aplikasi.

Berdasarkan informasi tersebut, subab mengenai Java ini sangatlah penting untuk dijadikan sebagai landasan teori dan referensi dalam penelitian. Karena pembangunan aplikasi dalam penelitian ini akan menggunakan bahasa pemrograman Java dan ditulis dan menghasilkan bentuk aplikasi Android.

2.5 Object Oriented Programing (OOP)

Setelah beberapa periode, kapasitas dan kemampuan komputer berkembang dengan pesat. Sejak saat itu, pengembangan aplikasi mulai menjadi kompleks. Sayangnya, tidak ada satu dari bahasa pemrograman yang cukup matang pada saat itu yang bisa digunakan untuk mengembangkan aplikasi kompleks secara efektif. Beberapa pertanyaan bermunculan, seperti bagaimana cara mencegah duplikasi kode? Atau bagaimana cara agar kode yang disusun bisa digunakan berulang kali? Untuk mengatasi masalah ini, para *programmer* mulai mencari solusi dengan cara memecah suatu permasalahan besar ke dalam permasalahan yang lebih kecil. Kunci dari solusi ini adalah jika kita bisa mengatasi masalah yang besar, seharusnya kita bisa mengatasi masalah yang lebih kecil [12].

Melalui permasalaha-permasalahan dan solusi yang telah dicoba untuk dilakukan tersebut, maka mulailah bermunculan solusi berupa paradigma pemrograman. Secara istilah paradigma sendiri adalah sebuah teori atau susunan solusi mengenai bagaimana sesuatu harus diselesaikan, dibuat, atau dipikirkan. Menurut Robert W. Floyd, seorang *computer scientist* menyatakan bahwa paradigma pemrograman adalah cara untuk memetakan apa artinya melakukan komputasi, dan bagaimana tugas yang ada di dalam komputer harus terstruktur dan terorganisir [11].

Salah satu paradigma pemrograman yang pertama populer adalah *Structured Programming.* Dengan paradigma ini, pengelolaan fungsi dapat dilakukan dengan mudah dan proses debuging program juga dapat dilakukan. Paradigma ini populer selama dua dekade. Selama waktu itu, perkembangan komputer sangat pesat dan mulailah bermunculan masalah mengenai paradigma ini karena proses pengembangan program menjadi lebih kompleks. Misalkan jika aplikasi yang dikembangkan menggunakan tipe data tertentu di beberapa fungsi. Kemudian selama pembaruan aplikasi, kita memerlukan perubahan tipe data. Untuk melakukan itu, kita perlu mengubah keseluruhan tipe data yang ada di beberapa fungsi secara menyeluruh, karenanya memerlukan banyak waktu untuk dihabiskan untuk mengatasinya.

Melihat masalah yang muncul tersebut, gagasan mengenai paradigma *Object Oriented Programming (OOP)* mulai muncul. Alan Curtis Kay bersama koleganya lah yang merupakan pengagas paradigma tersebut. Dalam paradigma OOP, sebuah program terdiri dari objek-objek yang saling berhubungan. Sebuah algoritma untuk menyelesaikan suatu masalah akan dibungkus di dalam objek. Suatu objek dapat berinteraksi dengan objek lainnya dengan memberikan perintah ke dalam objek lain. Ketika objek menerima perintah, lalu memberikan respon dengan mengeksekusi algoritma yang ada di dalamnya. Lebih jauh, paradigma ini dapat dilakukan dengan cara menulis program ke dalam kelas-kelas sekaligus menerapkan prinsip penting di dalamnya yaitu *Abstraction, Encapsulation, Inheritance, dan Polymorphism.* Berikut ini adalah penjelasan lebih dalam mengenai prinsip yang ada dalam paradigma OOP [12].

1. *Class* dan *Object* (Kelas dan Objek)

   Merupakan inti dari paradigma OOP. Kelas berperan sebagai cetak biru atau template untuk objek, sedangkan objek adalah instance dari kelas. Ibaratkan seorang arsitek membuat cetak biru suatu rumah, dan rumah yang telah jadi adalah objeknya atau hasil instance dari cetak biru rumah. Setiap objek memiliki *property* (sifat), *method* (kebiasaan, kemampuan), dan *identity* (identitas). Dalam bahasa pemrograman memiliki tipe data primitif seperti *int, bool, float.* Dikatakan primitif karena tipe data ini sudah didefinisikan bersamaan dengan bahasa pemrograman yang digunakan. Tetapi jika menggunakan OOP, kita bisa membuat tipe data buatan melalui hasil implementasi dari kelas menjadi sebuah objek.
2. *Encapsulation*

   Dalam OOP, *programmer* tidak diizinkan untuk membiarkan data pada variable berjalan dengan bebas di dalam sistem. Sebagai gantinya, setiap data dan fungsi akan dibungkus di dalam kelas. Seperti itulah enkapsulasi, tujuannya adalah untuk memberikan batasan ke dalam dua hal tadi, sehingga keduanya tidak bisa diakses dengan bebas di luar objek. Namun, kabar baiknya dalam OOP memberikan jenis-jenis enkapsulasi seperti *private, protected,* dan *public. Private* akan memberikan batasan suatu data hanya

dapat diakses di dalam objek dirinya sendiri. Sedangkan *protected* memberikan batasan suatu data hanya bisa di akses di dalam objek dirinya sendiri atau objek anaknya. Lalu, *public* akan memberikan menghilangkan batasan dari *private* dan *protected*, sehingga data akan bisa diakses langsung di luar objek. Penjelasan tentang objek bawahan akan dijelaskan selanjutnya dalam *Inheritance.*

3. *Abstraction* (Abstraksi)

   Tujuan dari abstraksi adalah untuk menyembunyikan suatu kode program yang kompleks dan menjadikannya lebih sederhana. Contoh di kehidupan nyata adalah ketika menyalakan televisi melalui *remote*, kita hanya akan tahu setelah *remote* ditekan maka televisi akan menyala dan kita tidak perlu tahu mekanisme apa yang dijalankan di dalamnya untuk bisa menyalakan televisi tersebut. Contoh konsep abstraksi ini masihlah sangat luas, contoh tadi hanya perumpamaan agar konsep ini bisa dipahami dengan mudah.

4. *Inheritance* (Pewarisan)

   Merupakan sebuah proses agar sebuah objek bisa memiliki kemampuan objek lain. Konsep ini sangat berguna jika berbicara tentang reusability. Contoh sederhananya adalah misalkan kita memiliki sebuah *remote* yang bisa untuk menyalakan sesuatu, kemudian kita memiliki televisi dan radio yang ingin bisa dinyalakan melalui *remote*. Maka dari itu, dibuatlah *remote* televisi dan *remote* radio. Sehingga dapat dikatakan bahwa kedua *remote* baru itu hasil bentukan dari *remote* utama tadi. Istilah dalam OOP, *remote* utama adalah parent dan kedua *remote* yang baru itu adalah *child* atau anak dari *remote* utama.

5. *Polymorphism* (Polimorfisme)

   Konsep polimorfisme adalah satu nama dengan banyak bentuk. Dalam pemrograman, polimorfisme adalah kemampuan dari suatu objek untuk bisa digunakan dalam tujuan yang berbeda. Contohnya adalah kita memiliki suatu objek bernama BangunDatar dengan method hitungLuas(). Method tersebut dapat kita gunakan untuk menghitung luas bangun datar apapun. Untuk melakukan hal ini maka konsep polimorfisme ini perlu diterapkan. Dengan

konsep ini satu metode dengan nama yang sama dapat memiliki kemampuan atau bentuk yang berbeda-beda.

Pengentahuan tentang paradigma pemrograman berbasis objek atau OOP ini sangat membantu dalam proses penelitian terutama saat membangun aplikasi yang dijadikan sebagai hasil dan tujuan dari penelitian ini. Maka dari itu, landasan teori mengenai OOP penting untuk dicantumkan di sini.

2.6 Unified Modelling Language (UML)

Unified Modelling Language (UML) merupakan standar bahasa modeling untuk pengembangan software dan sistem. Ketika mengembangkan software berskala besar dan kompleks akan sangat sulit untuk dikelola dan dipahami terutama jik dikerjakan sebagai tim. Tidak hanya itu, bagaimanapun software yang dikembangkan itu terbilang terbilang sederhana ataupun kompleks. Keduanya akan sangat mungkin terdiri dari puluhan hingga ratusan bahkan ribuan komponen kecil. Ketika mencapai tahap ini, bagaimana selaku pengelola proyek bisa menjamin untuk bisa menjelaskan dengan baik alur kerja software atau sistem yang dikembangkan kepada *programmer* atau kolega yang menjalin kerjasama dalam pengembangan software tersebut? Akan sangat lama dan lambat untuk bisa menjelaskan satu per satu komponen yang ada di dalam software tersebut. Maka dari itu, diterapkanlah bahasa standar modeling bernama UML [13].

Alasan lain menerapkn UML sebagai standar modeling adalah krena UML dikembangkan dengan kelebihannya adalah :

1. Menggunakan Bahasa Formal

   Setiap elemen didefinisikan dengan bahasa formal yang sesuai dengan maksud dan tujuan dari elemen tersebut. Dengan bahasa formal, setiap orang akan mudah untuk memahami apa maksud dari elemen-elemen tersebut.
2. Ringkas

   Setiap bahasa pemodelan disusun dengan ringkas dan sederhana sesuai dengan tujuan dari setiap elemen.

3. Komprehensif

   Hanya memodelkan elemen yang penting yang ikut berperan di dalam sistem.

4. Terukur

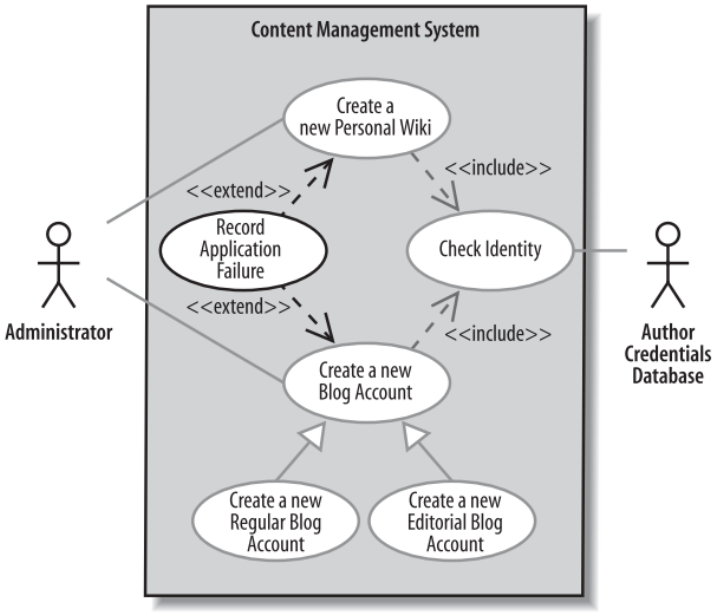
   UML memiliki kemampuan untuk dapat memodelkan suatu sistem yang besar dan kompleks, namun terkadang bisa juga disesuaikan kembali ukurannya ke dalam bagian-bagian kecil dari suatu sistem yang besar.

Dalam melakukan modeling menggunakan UML akan terdiri dari susunan elemen-elemen. Setiap elemen tersebut diberi istilah notation atau simbol. Hal ini dilakukan agar setiap orang dapat memahami dengan mudah elemen apa yang sedang dimodelkan.

Untuk membedakan konteks dari setiap notation yang ada, maka UML membaginya kembali ke dalam jenis diagram. Berikut ini adalah diagram yang disediakan dalam standar UML.

2.6.1 *Use Case Diagram*

Bayangkan jika saat membuat sistem, misalkan sistem pengelolaan penjualan. Jika pengembang langsung mengembangkan sistem tanpa ada perencanaan fitur, kebutuhan, dan pengguna di dalamnya, maka dapat ditebak bahwa pengembangan sistem tersebut akan tidak terarah dan tidak diketahui hasil akhir dari sistem tersebut akan menjadi seperti apa. Untuk mengatasi masalah ini, UML memberikan solusi dengan adanya *Use Case Diagram. Use Case* adalah sebuah situasi atau *case* di dalam sistem yang digunakan untuk mengisi satu dari sekian banyak kebutuhan sistem. *Use Case* memodelkan setiap fungsi yang ada di dalam sistem dan menentukan siapa saja yang akan terlibat di dalamnya.

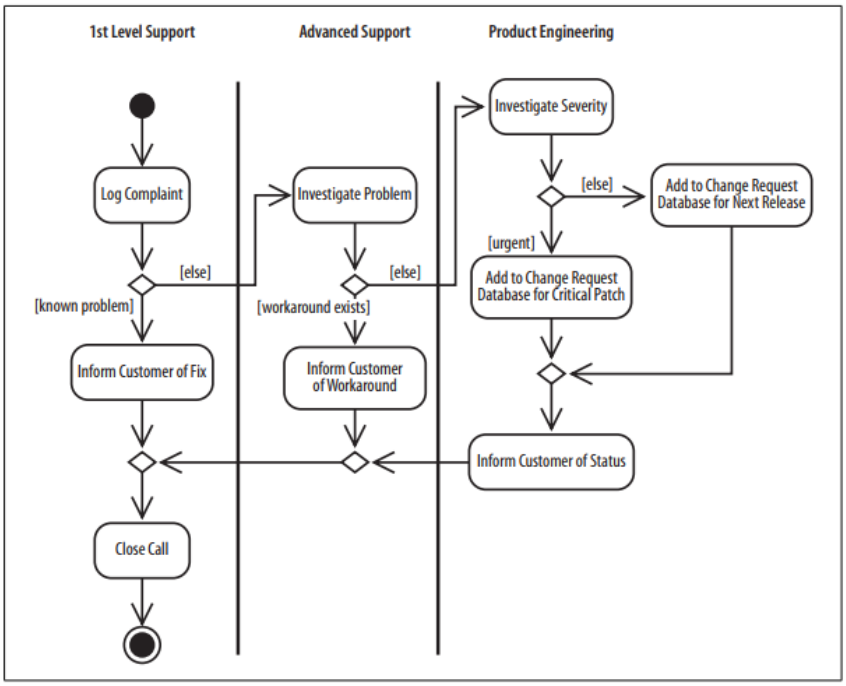


Gambar 2.8 Contoh *Use Case Diagram*

Pada Gambar 2.8 terlihat dua bagian penting yaitu Aktor sebagai pengguna, dan *Use Case* sebagai fungsi-fungsi di dalam ssitem. *Use Case* memodelkan kebutuhan sistem secara ketat dan dapat menentukan keubuthan setiap pengguna. Karena *Use Case* adalah model dari kebutuhan fungsional dalam sistem, maka ini harus diberikan perhatian khusus di awal sebelum lanjut ke tahap pemodelan lainnya. Setelah merancang kebtuhan fungsional sistem, selanjutnya adalah menyusun skenario yang ada di dalam setiap *Use Case*, skenario ini menentukan kondisi awal sampai akhir yang harus ada dan dicapai oleh suatu *Use Case.* Sehingga dapat diketahui bahwa *Use Case Diagram* ini berperan untuk menentukan apa yang sistem dapat lakukan.

2.6.2 *Activity Diagram*

Setelah memodelkan dan mendefinisikan kebutuhan fungsional sistem, selanjutnya adalah bagaimana seseorang dapat mengetahui susunan aktifitas yang dilakukan dalam suatu *Use Case* dari awal keadaan terjadi hingga mencapai tujuan akhirnya. Solusi dari masalah ini adalah dengan menggunakan *Activity Diagram.*



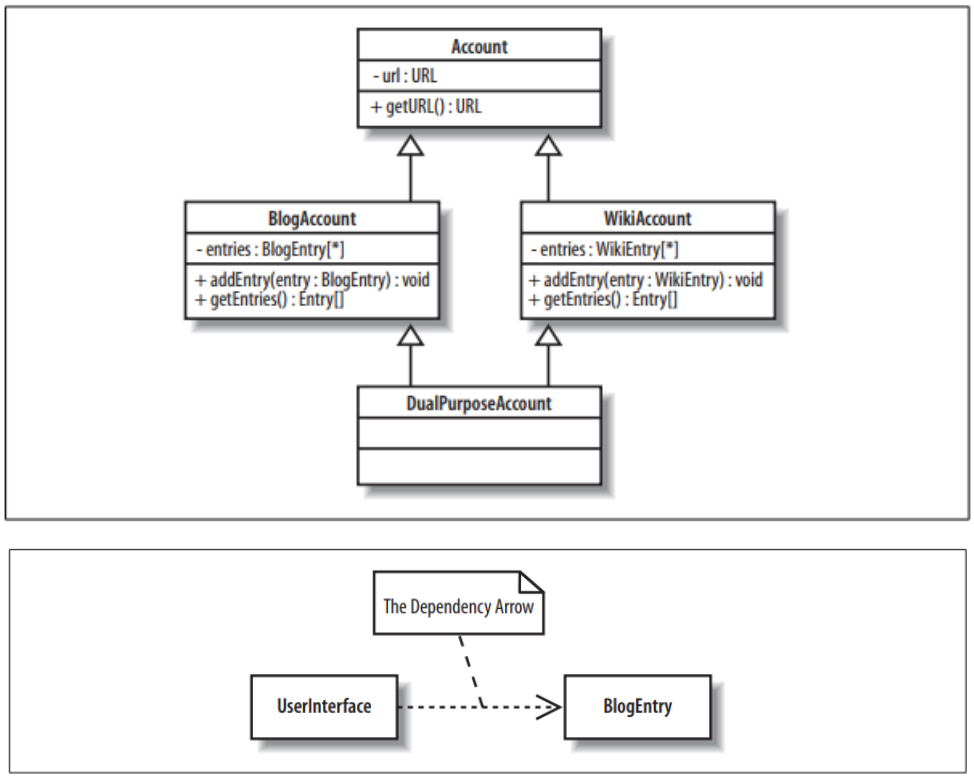
Gambar 2.9 Contoh *Activity Diagram*

Activity Diagram digunakan untuk memodelkan susunan aktifitas dari suatu *Use Case*. Seperti memperlihatkan cara kerja suatu *Use Case* dari titik awal sampai ke titik akhir secara mendetail. Sebagai contoh kita dapat menggunakan *Activity Diagram* untuk menggambarkan aktifitas yang harus dilalui dalam membuat artikel *blog*. Melihat dari definisi dan contoh, dapat diketahui bahwa *Activity Diagram* secara praktis dan detail cocok untuk memodelkan aturan bisnis yang ada di dalam suatu sistem.

2.6.3 Class Diagram

Dalam paradigma pemrograman berbasis objek (OOP), *Class* (Kelas) merupakan intinya. Setiap permasalahan yang ada dalam sistem akan dipecahkan melalui susunan berbagai kelas yang saling terhubung. Di dalam satu kelas terdapat atribut dan *method* yang menjadi penyusun untuk mencapai tujuan dari suatu kelas. Dalam sistem yang menerapkan OOP, baik itu kompleks maupun sederhana pasti memiliki banyak kelas di dalamnya. Jika tidak diatur keberadaan setiap kelas, maka sistem tersebut akan sulit untuk dikelola dan dikembangkan ke depannya karena tidak ada suatu penggambaran khusus akan peran dan relasi

setiap kelas yang ada. Untuk menjawab masalah ini, UML menyediakan solusi berupa *Class Diagram.*



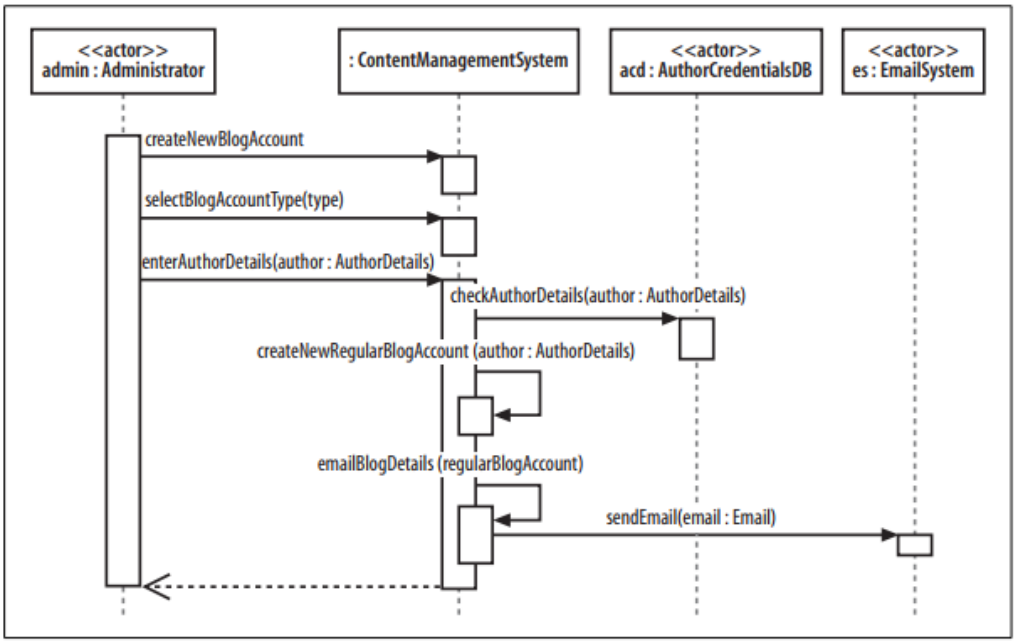
Gambar 2.10 Contoh *Class Diagram*

Class Diagram sangat berguna untuk mengilustrasikan hubungan antar kelas di dalam sistem. Hubungan-hubungan yang diakomodasi oleh *Class Diagram* seperti *Generalization, Aggregation,* dan *Association.* Hal ini dapat dilihat pada Gambar 2.10 di atas. Dalam contoh diagram di atas diperlihatkan bagaimana setiap kelas dalam suatu sistem saling berhubungan satu sama lain.

2.6.4 Sequence Diagram

Setelah merancang dan memodelkan kelas yang terlibat di dalam sistem, permasalahan selanjutnya adalah bagaimana cara mengetahui alur peranan dari setiap kelas yang ada sehingga dapat menyelesaikan suatu permasalahan di dalam sistem? Untuk menjawab ini, maka UML memberikan solusi berupa *Sequence Diagram. Sequence Diagram* adalah diagram yang memperlihatkan interaksi antar kelas di dalam sistem dalam bentuk garis waktu, dimulai dari atas yang

menandakan awal interaksi terjadi sampai ke bawah yang menandakan akhir dari interaksi.



Gambar 2.11 Contoh *Sequence Diagram*

Sequence Diagram memperlihatkan setiap aktifitas secara berurutan, memungkinkan setiap orang tahu bagaimana setiap objek berkomunikasi satu sama lain hingga mencapai tujuan akhir. Gambar 2.11 memperlihatkan setiap objek memiliki garis waktu masing-masing, setiap interaksi antar objek disebut *message* yang ditunjukkan oleh anak panah. Interaksi ini akan terus terjadi sampai tujuan tercapai.

Subab mengenai UML ini sangat penting untuk dicantumkan, karena jika dilihat dari setiap diagram yang ada pada UML, mengacu pada paradigma pemrograman berbasis objek. Hal ini sejalan dengan batasan dari penelitian ini, di mana proses pembangunan aplikasi akan menerapkan paradigma berbasis objek.

2.7 Application Programming Interface (API)

Application Programming Interface (API) secara umum memiliki definisi sebagai sebuah cara bagaimana sistem komputer berinteraksi. Sekarang, API telah ada di dalam banyak sistem, baik itu dalam *software*, bahasa pemrograman, *library*, dan sebagainya. Sebagai contoh, sebuah software memberikan API untuk

dapat melakukan enkripsi teks [14]. Dengan API, setiap orang bisa melakukan enkripsi hanya melalui fungsi yang diberikan dan dapat dipanggil di luar software. Contoh tersebut adalah API yang ada di dalam *software.*

Sedangkan, di dalam ekosistem software ada banyak jenis API. salah satu jenis API yang spesial adalah yang bisa diakses oleh setiap orang melalui jaringan. API diakses melalui alamat *Uniform Resource Locators (URLs)* yang telah disediakan. Dengan mengakses alamat itu, seseorang bisa menambah, mengubah, dan menghapus data melalui jaringan internet tanpa harus berinteraksi langsung ke dalam sistem. API ini dinamakan sebagai Web API.

Penggunaan Web API di dalam pengembangan aplikasi masa kini sangat penting, apalagi internet yang sudah demikian pesat berkembang. Dapat dibayangkan jika *programmer* mengembangkan aplikasi smartphone untuk jual beli barang. Bagaimana data yang ditampilkan di dalam aplikasi akan didapatkan dan disimpan, sedangkan data tersebut ada di luar aplikasi yang dipasang. Untuk bisa melakukan itu, maka *programmer* akan menggunakan API yang berperan sebagai jalur untuk komunikasi data antar sistem.

Landasan teori mengenai API pada subab ini sangat penting untuk dicantumkan. Karena dalam penelitian ini, komunikasi data tidak hanya terjadi di dalam aplikasi, tetapi memerlukan komunikasi data di luar aplikasi, seperti mendapatkan data lokasi Wi-Fi dan menyimpan data penayangan iklan.

2.7.1 Google Maps API

Google Maps pertamakali diperkenalkan oleh Google pada Februari 2005. Ketika pertama kali diperkenalkan, Google Maps belum memiliki API secara publik. Hingga, beberapa pengembang menemukan cara untuk meretas dan menyisipkan Google Maps ke dalam situs di luar Google Maps. Melihat alasan ini, Google menyadari bahwa adanya kebutuhan terhadap API publik. Kemudian di bulan Juni 2005, Google merilis API secara publik. Halaman depan Google Map dibangun atas HTML, CSS, dan JavaScript. Setiap gambar peta di dalam situs, terdiri dari bagian-bagian kecil gambar. Setiap gambar dimuat melalui

AJAX ke API Google Maps. Setiap kali pengguna melakukan navigasi pada peta, maka bagian-bagian peta baru akan muncul dengan melakukan hal yang sama melalui AJAX [15].

Google Maps menyediakan berbagai jenis API yang dapat digunakan untuk segala kebutuhan terkait dengan lokasi dan peta. Jika pengembang ingin mendapatkan informasi suatu tempat dan koordinat, pengembang dapat menggunakan Places API. Dengan API ini pengembang dapat memanfaatkan fitur Geocoding dan Geolocation. Geocoding memiliki kemampuan untuk mendapatkan alamat berdasarkan koordinat, dan Geolocation dapat digunakan untuk mendapatkan koordinat perangkat berdasarkan tower telepon dan atau *Node* Wi-Fi. Selain Places API, Google Maps juga menyediakan API berbentuk *library*. *Library* ini dapat digunakan untuk pengembangan di berbagai platform seperti Web dan Android. API ini sering dinamakan *Software Development Kit (SDK).* Dengan menggunakan API jenis ini, pengembang dapat memasukan peta ke dalam aplikasi dan mengubah tampilan peta di dalamnya.

Di dalam penelitian ini, Places API dan SDK sangat penting untuk digunakan. Karena dalam pembangunan aplikasi Wi-Fi, akan melibatkan koordinat dan lokasi pengguna pada saat menggunakan aplikasi. Selain itu, penggunaan Google Maps SDK juga sangat penting untuk digunakan agar aplikasi yang diteliti dapat ditambahkan peta di dalamnya. Sehingga, subab mengenai Google Maps API ini perlu untuk dicantumkan sebagai referensi pada saat dilakukan penelitian dan pengembangan.

2.7.2 Google AdMob API

Google AdMob adalah salah satu penyedia layanan jaringan iklan daring dari Google. Membantu pengembang memonetisasi aplikasi dengan menayangkan iklan di dalam aplikasi. Iklan dapat ditampilkan dalam berbagai format, seperti *banner ads, interstitial ads, native ads,* dan *video ads.* Untuk dapat mulai memonetisasi aplikasi, pengembang harus membuat akun AdMob terlebih dahulu. Setelah itu baru pengembang dapat membuat unit iklan baru dan mendapatkan ID untuk dapat dijadikan acuan penayangan iklan di dalam aplikasi [16].

Untuk membantu pengembang memaksimalkan kemampuan dalam monetisasi aplikasi, Google AdMob memberikan akses ke dalam akun AdMob melalui API. API yang diberikan oleh AdMob berupa *library* yang dapat digunakan diberbagai bahasa pemrograman, istilah untuk *library* ini sering disebut sebagai *Software Development Kit (SDK).* Melalui SDK ini, pengembang bisa mendapatkan berbagai informasi seperti melihat estimasi pendapatan, dan melihat daftar unit iklan.

Subab mengenai API yang disediakan oleh Google AdMob ini penting untuk dicantumkan sebagai landasan teori penelitian. Karena dalam penelitian pembangunan aplikasi berbagi Wi-Fi ini akan menayangkan iklan yang diambil dari Google AdMob serta bisa mendapatkan laporan harian estimasi pendapatan dari iklan yang ditampilkan.

2.8 Javascript Object Notation (JSON)

JavaScript Object Notation (JSON) merupakan *subset* atau himpunan bagian dari Bahasa pemrograman JavaScript. Sehingga pada awalnya JSON menjadi bagian dari JavaScript. Sebagai turunan, JSON tidak mewarisi kemampuan seperti JavaScript. Secara spesifiknya, JSON hanyalah sebuah subset dari JavaScript dan bukan sebagai bahasa pemrograman, tapi sebagai *Data Interchange Format* (Format Pertukaran Data) [17].

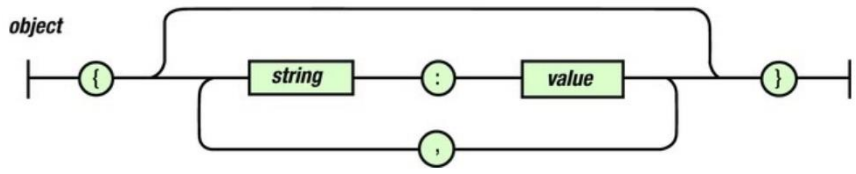
Format pertukaran data adalah format teks yang digunakan untuk pertukaran data antar platform. Ada format pertukaran data yang cukup popular sebelum JSON, yaitu XML yang ditulis dengan format Bahasa *Markup.* Dunia pemrograman membutuhkan format pertukaran data yang keberadaannya dapat membantu proses pertukaran data antar sistem. Dapat dibayangkan jika di dunia pemrograman tidak memiliki standar format pertukaran data, akan sangat sulit untuk sistem dapat berkomunikasi dengan sistem yang lainnya. JSON adalah salah satu format pertukaran data yang telah digunakan oleh banyak sistem dalam melakukan komunikasi data [18].

JSON secara dasar hanyalah sebuah representasi tekstual yang menggunakan suatu aturan sederhana sehingga dapat terstruktur dan dipahami oleh semua kalangan. Spesifikasi aturan penulisan JSON sangat sederhana, bahkan hanya memiliki dua komposisi format penulisan. Yaitu penulisan *collection (object)* dengan menggunakan pasangan *name* dan *value*, serta penulisan *list (array)* yang dapat berisi *value*, hal ini sesuai dengan standar ECMAScript yang menyatakan bahwa implementasi dari kedua komposisi penulisan JSON direpresentasikan sebagai bentuk dari *Object* dan *Array.*

Agar lebih jelas, berikut ini adalah komposisi penulisan dan tipe data yang didukung dalam JSON :

1. *Collection (Object)*

   JSON diharuskan untuk menulis *collection* dengan format seperti pada diagram berikut.



Gambar 2.12 Diagram Penulisan *Object* di JSON

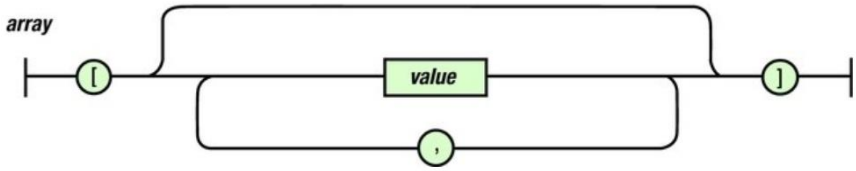
Gambar 2.12 memberikan ilustrasi bahwa suatu collection di dalam JSON harus didefinisikan dengan diawali oleh kurung kurawal buka ({) dan diakhiri oleh kurung kurawal tutup (}). Setiap kali *collection* didefinisikan, disebut sebagai sebuah blok *collection.* Di dalam blok dapat berisi data yang ditulis dengan format *name/value.* Di mana *name* harus berupa *string* dan ditulis di dalam *quote-unquote* ("…"). Sedangkan *value* yang merupakan nilai dari data, dapat berupa *collection, list, string, number,* atau *boolean.* Suatu blok dapat memiliki banyak pasangan name/value dengan cara dipisahkan oleh tanda koma (,). Semua ini sesuai dengan ilustrasi yang ada pada diagram di atas. Berikut ini adalah contoh penulisan *collection* secara tekstual.

```
{
    "data": {
        "name": "Satria Aji",
        "departement": "Teknik Informatika",
        "nim": "10118068",
        "age": 22,
        "has_graduated": false,
        "hobbies": ["Eat", "Learning"]
    }
}
```

Gambar 2.13 Contoh Penulisan Collection JSON

2. *List (Array)*

   JSON diharuskan untuk menulis list dengan menggunakan format seperti pada diagram berikut.



Gambar 2.14 Diagram Penulisan *Array* di JSON

Gambar 2.14 memberikan ilustrasi format penulisan array di JSON yang harus dilakukan dengan diawali oleh kurung siku buka ([) dan diakhiri dengan tanda kurung siku tutup (]). Setiap kali list didefinisikan, disebut sebagai blok list. Di dalam blok list dapat berisi satu atau lebih value. Value yang dapat ditambahkan berjenis collection, array, string, boolean, dan number. Jika value yang ditambahkan lebih dari satu, maka disetiap akhir value harus ditambahkan tanda koma (,). Berikut ini adalah contoh penulisan blok array di JSON.

```
{
    "validation_messages": [
        "Name is required",
        "Age is required",
        "Department is required"
    ],
    "points": [20,30,90]
}
```

Gambar 2.15 Contoh Penulisan *List* di JSON

3. *String*

   String adalah salah satu tipe data yang didukung dalam format pertukaran data JSON. Penulisan string dapat dilakukan di dalam tanda *quote-unquote* ("…"). Jika di dalam tanda tersebut terdapat *string* yang mengandung *quote*, maka harus dilakukan *parsing* dengan cara memberi tanda garis miring (\) di sebelum *quote*. Contoh "Tempat itu dinamakan \"salatiga\", berada di…".

4. *Number*

   Tipe data yang didukung selanjutnya adalah *Number*. Tipe data ini dapat ditulis secara langsung tanpa harus ada tanda *quote* seperti pada *String*. *Number* dapat berupa bilangan bulat maupun bilangan desimal. Untuk bilangan desimal, tanda koma harus digantikan dengan tanda titik.

5. *Boolean*

   Tipe data terakhir adalah *Boolean*, merupakan tipe data yang hanya memiliki dua nilai, yaitu *true* dan *false*. Penulisan *boolean* di JSON dapat ditulis secara langsung tanpa harus ada tanda *quote*, yaitu langsung menuliskan *true* atau *false*.

2.9 MariaDB

Di masa perkembangan teknologi yang pesat saat ini, database berada di sekitar kita. Hampir setiap *website* dan aplikasi yang orang gunakan disetiap harinya menggunakan database di belakang layar. Semua informasi yang disimpan dapat memiliki bentuk yang berbeda-beda. Seperti katalog buku, alamat pengguna, katalog produk, keuangan, katalog obat, dan lain sebagainya. Semua

jenis informasi itu disimpan di dalam komputer secara elektronis. Dengan melihat kegunaan database seperti itu, menjadikannya sebagai salah satu pilar fundamental yang penting keberadaannya di dalam pengembangan aplikasi.

Jenis-jenis database seperti MySQL, Oracle, SQLite telah menjadi populer untuk waktu yang lama. Namun sekitar tahun 2009, ada jenis database baru yang cukup menarik perhatian pegiat komunitas database di dunia. Nama dari database itu adalah MariaDB. Diciptakan oleh Ulf Michael Widenius dan biasa dipanggil dengan Monty. Fakta menarik bahwa Monty adalah salah seorang yang ikut berkontribusi besar dalam menciptakan MySQL. Meski MariaDB ini terbilang masih muda, tapi asal-usulnya tidak dapat diragukan. Karena database MariaDB merupakan generasi berikutnya dari database MySQL yang sejak lama telah populer.

MariaDB merupakan jenis database yang sumbernya terbuka *(open source)*. Setiap orang dapat dengan bebas mengunduh kode sumber dan dapat mengembangkannya jika mereka mau. Sejak pertama rilis, MariaDB telah mendapat sambutan yang besar dari para pengembang perangkat lunak, terutama pengembang *website*. Sejak tahun 2015, MariaDB telah digunakan oleh puluhan ribu *website*, besar dan kecil, dan jenis database ini juga digunakan oleh banyak perusahaan dengan latar belakang industri yang berbeda-beda di seluruh dunia dan digunakan oleh ratusan-ribu pengguna [19].

Meski latar belakang MariaDB berasal dari MySQL, namun perbedaan paling menonjol dan menjadi salah satu poin perhatian adalah keterbukaan kode sumber MariaDB. Dengan jenis lisensi seperti ini, setiap orang dapat dengan bebas menggunakan database jenis ini untuk segala macam kebutuhan penggunaan. Sedangkan MySQL yang saat ini berada di bawah Oracle, sejak lama menggunakan lisensi *General Public License (GPL)*. Dengan lisensi jenis ini, MySQL bebas digunakan, akan tetapi ketika digunakan dalam aplikasi yang bersifat komersil dan dapat menghasilkan pendapatan di dalamnya, pengembang wajib membeli lisensi komersil ke pihak Oracle [20].

Meskipun dengan lisensi sumber terbuka, MariaDB dapat mendukung dan dijalankan di berbagai jenis sistem operasi. Bahkan saking terbukanya sumber, setiap orang dapat melakukan build ulang MariaDB dan menjadikannya suatu installer baru serta dapat dijalankan di suatu sistem operasi yang baru. Saat ini, MariaDB memiliki paket instalasi perangkat lunak yang mendukung di beberapa jenis sistem operasi seeperti Windows, Linux, dan macOS.

Maka dari itu, penelitian yang dilakukan saat ini akan memilih MariaDB sebagai *Database Management System (DBMS)* utama yang akan digunakan dalam pembangunan aplikasi. Aplikasi akan menggunakan MariaDB utntuk menyimpan segala jenis informasi yang dihasilkan dari aplikasi dan didapatkan melalui pengguna. Sehingga subab mengenai MariaDB ini penting untuk dicantumkan sebagai landasan teori.