BAB II

TINJAUAN PUSTAKA

Dalam BAB ini berisi berbagai konsep dasar dan teori-teori yang berkaitan dengan Pembangunan Aplikasi Chatbot untuk Diagnosa Penyakit pada Ikan Lele Menggunakan Metode Pengolahan Citra.

2.1 Line

Line merupakan *platform* komunikasi yang sangat diminati oleh banyak orang terutama dikalangan remaja. Hingga saat ini, dari total 220 juta penggunanya di seluruh dunia, sebanyak 90 juta di antaranya berasal dari Indonesia. Jumlah pengguna ini terbesar keempat setelah Jepang, Taiwan, dan Thailand. LINE juga memiliki Line *Chatbot* yang memungkinkan Anda untuk membuat *chat* berbasis bot yang dapat dijalankan di *platform* Line [7].

2.2 Opencv

OpenCV (*Open Source Computer Vision Library*), adalah sebuah *library open source* yang dikembangkan oleh *intel* yang fokus untuk menyederhanakan programing terkait citra digital. Pada *OpenCV* sudah mempunyai banyak fitur, antara lain : pengenalan wajah, pelacakan wajah, deteksi wajah, Kalman filtering, dan berbagai jenis metode AI (*Artificial Intellegence*). Dan menyediakan berbagai algoritma sederhana terkait *Computer Vision* untuk *low level API* [8].

2.3 Webhook

Sebuah *webhook* dalam pengembangan *web* adalah metode untuk menambah atau mengubah perilaku halaman *web*, atau aplikasi *web*, dengan *callback* kustom. *Callback* ini dapat dipertahankan, dimodifikasi, dan dikelola oleh pengguna dan pengembang pihak ketiga yang mungkin tidak perlu berafiliasi dengan situs *web* atau aplikasi yang berasal [10].

2.4 Ikan Lele

Lele atau ikan keli, adalah sejenis ikan yang hidup di air tawar. Lele mudah dikenali karena tubuhnya yang licin, agak pipih memanjang, serta

memiliki "kumis" yang panjang, yang mencuat dari sekitar bagian mulutnya [11].

2.5 Phyton

Python merupakan bahasa pemrograman yang populer dan banyak digunakan oleh Data *Analysts*, Data *Scientists* dan para *Software Engineers* untuk menjalankan proses pembangunan sebuah aplikasi dan untuk menggali lebih dalam *machine learning* [12].

2.6 Aplikasi

Aplikasi adalah suatu perangkat lunak (*software*) atau program komputer yang beroperasi pada sistem tertentu, yang diciptakan dan dikembangkan untuk melakukan perintah tertentu. Istilah aplikasi sendiri diambil dari bahasa Inggris "*application*" yang dapat diartikan sebagai penerapan atau penggunaan. Secara harfiah, aplikasi merupakan suatu penerapan perangkat lunak atau *software* yang dikembangkan untuk tujuan melakukan tugas-tugas tertentu. [13]

Dalam pengembangannya, aplikasi dapat dikategorikan dalam tiga kelompok, diantaranya; Aplikasi desktop, yaitu aplikasi yang hanya dijalankan di perangkat komputer atau laptop. Lalu aplikasi *web*, yaitu aplikasi yang dijalankan menggunakan komputer dan koneksi internet. Dan aplikasi *mobile*, yaitu aplikasi yang dijalankan di perangkat *mobile* di mana untuk kategori ini penggunaannya sudah banyak sekali.

Umumnya suatu aplikasi dapat berjalan di berbagai perangkat yang dioperasikan oleh *operating system* (*OS*) yang ada di perangkat tersebut. Aplikasi juga dapat memberikan kemudahan dan kenyamanan bagi manusia di berbagai bidang kehidupan, seperti: Pendidikan, Kedokteran, Bisnis, Ilmu pengetahuan, dan juga Militer.

2.7 Object Oriented Programming

Pemrograman berorientasi objek adalah paradigma pemrograman yang didasarkan pada konsep "objek", yang berisi data, dalam bentuk *field* atau dikenal juga sebagai atribut; serta kode, dalam bentuk fungsi atau prosedur atau dikenal juga sebagai *method*. Semua data dan fungsi pada paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Bandingkan

dengan logika pemrograman terstruktur. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya [14].

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik peranti lunak skala besar. Lebih jauh lagi, pendukung *OOP* mengklaim bahwa *OOP* lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan *OOP* lebih mudah dikembangkan dan dirawat.

2.7.1 Objek

Objek adalah pembungkus data dan berfungsi bersama menjadi satu kesatuan suatu unit dalam sebuah program komputer; objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

2.7.2 Kelas

Kelas merupakan kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *'class of dog'* adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. Sebuah *class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi object. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang *non-programmer* sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah *class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan *OOP*). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

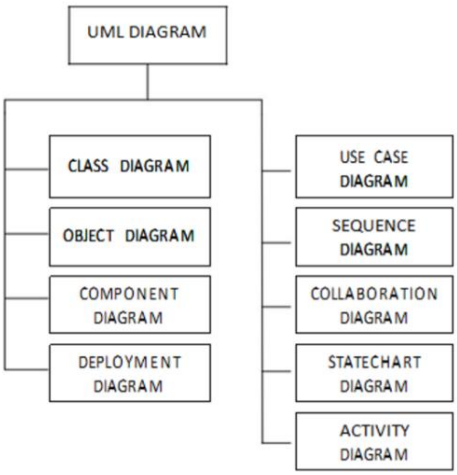
2.8 UML

Unified Modeling Language (UML) merupakan kumpulan struktur ataupun teknik yang digunakan untuk memodelkan desain program

berorientasi objek (OOP). UML merupakan metode yang digunakan untuk mengembangkan sistem program berorientasi objek (OOP) dan sekelompok perangkat tool untuk pengembangan sistem tersebut. UML diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP.

2.8.1 UML Diagram

Diagram-diagram yang terdapat pada UML sangat banyak, berikut ini beberapa diagram yang sering digunakan dalam pengembangan sistem dapat dilihat pada gambar dibawah .



Gambar 2.1 UML Diagram

2.8.2 Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut(metoda/fungsi) [15]. Class diagram menggambarkan struktur dan deskripsi class, *package* dan objek beserta hubungan satu sama lain seperti containment, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok yaitu Nama (dan *stereotype*), Atribut dan Metoda. Atribut dan metoda dapat memiliki salah satu sifat berikut :

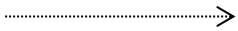
1. *Private* (-), tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected* (#), hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public* (+), dapat dipanggil oleh siapa saja.

Hubungan Antar *Class* :

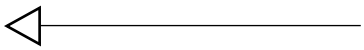
1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*. Lambang :

2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..") atau biasa disebut relasi mempunyai sebuah. Lambang :
3. *Composition*, yaitu sebuah kelas tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* yang lain, maka *class* tersebut memiliki relasi *composition*. Lambang:

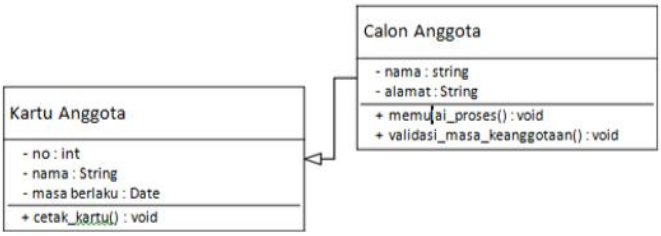
4. *Dependency*, yaitu hubungan yang menunjukan operasi pada suatu kelas yang menggunakan *class* yang lain. Lambang :



5. Generalisasi / Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwaris.



Berikut Contoh *Class* Diagram dapat dilihat pada Gambar 2.2.

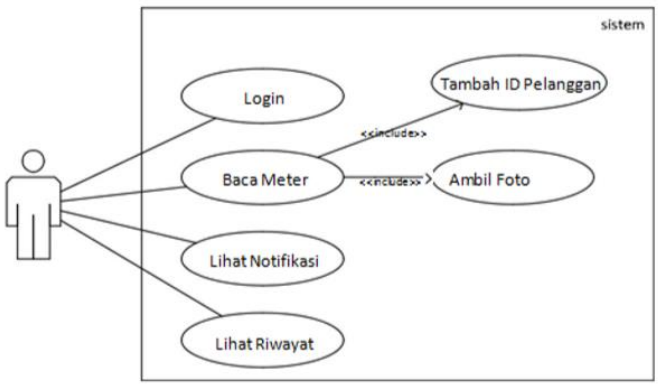


Gambar 2.2 Class Diagram

2.8.3 Use Case Diagram

Use Case Diagram yaitu diagram yang digunakan untuk menggambarkan hubungan antara sistem dengan aktor. Diagram ini hanya menggambarkan secara global. Karena use case diagram hanya menggambarkan sistem secara global, maka elemen-elemen yang digunakan pun sangat sedikit, berikut ini elemen-elemen yang digunakan pada *use case* diagram :

1. Sistem, merupakan batasan-batasan proses yang sudah kita deskripsikan dałam sebuah sistem.
2. Aktor, elemen yang menjadi pemicu sistem. Aktor bisa berupa orang, mesin ataupun sistem lain Yang berinteraksi dengan *use case*.
3. *Use Case*, potongan proses Yang merupakan bagian dari sistem.
4. *Association*, menggambarkan interaksi antara *use case* dan aktor.
5. *Depedency*, menggambarkan relasi (*relationship*) antara dua *use case*. Ada 2 (dua) tipe dari *dependency* yaitu, *include* dan *extends*. *Include* merupakan tipe dari *dependency* yang menghubungkan dua *use case* dimana, satu *use case* membutuhkan *use case* yang satunya sedangkan *extends* adalah tipe dari *dependency* yang menghubungkan dua *use case* dimana satu *use case* terkadang akan memanggil *use case* yang satunya, tergantung pada kondisi.
6. *Generalization*, menggambarkan pewarisan antara dua aktor atau *use case* dimana salah satu aktor atau *use case* mewarisi *properties* ke aktor atau *use case* yang satunya. Contoh *Use Case* Diagram dapat dilihat pada Gambar 2.3.



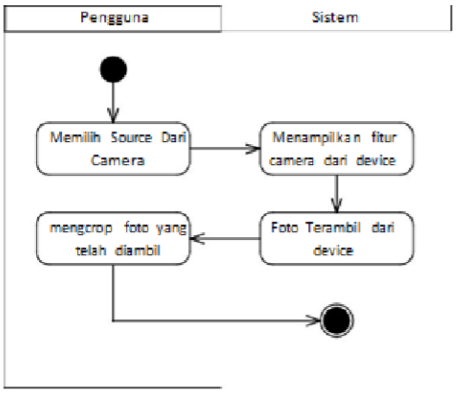
Gambar 2.3 Use Case Diagram

2.8.4 Activity Diagram

Activity Diagram yaitu diagram yang digunakan untuk menggambarkan alur kerja (aktivitas) pada *use case* (proses), logika, proses bisnis dan hubungan antara aktor dengan alur-alur kerja *use case*. Jika sudah terbiasa dengan *flowchart*, maka tidak akan merasa kesulitan dalam mempelajari *activity* diagram, karena *activity* diagram identik dengan *flowchart*, hanya saja ada beberapa notasi tambahan yang digunakan untuk kasus- kasus tertentu. Berikut ini di jelaskan elemen-elemen dari *activity* diagram.

1. *Activity*, yaitu elemen yang digunakan untuk menggambarkan aktivitas.
2. *Transitions*, yaitu elemen yang digunakan untuk menggambarkan transisi dari elemen yang satu ke elemen yang lainnya
3. *Decisions*, yaitu elemen yang digunakan untuk percabangan logika. Elemen ini sering kita jumpai pada *flowchart* terutama *flowchart* yang digunakan untuk menggambarkan sebuah algoritma.
4. *Merge Point*, yaitu elemen yang digunakan untuk menggabungkan percabangan proses. Elemen ini merupakan kebalikan dari elemen *decisions*, dimana jika *decisions* digunakan untuk percabangan, sedangkan *merge point* digunakan sebagai penggabungan dari percabangan.
5. *Start Point*, yaitu elemen yang digunakan untuk memulai *activity* diagram.
6. *End Point*, yaitu elemen yang digunakan untuk mengakhiri *activity* diagram.
7. *SwimLines*, yaitu elemen yang digunakan untuk memisahkan antara aktor dan sistem ataupun antara aktor yang satu dengan aktor yang lain atau antara sistem yang satu dengan sistem yang lain.

Berikut Contoh *Activity* Diagram dapat dilihat pada gambar 2.4



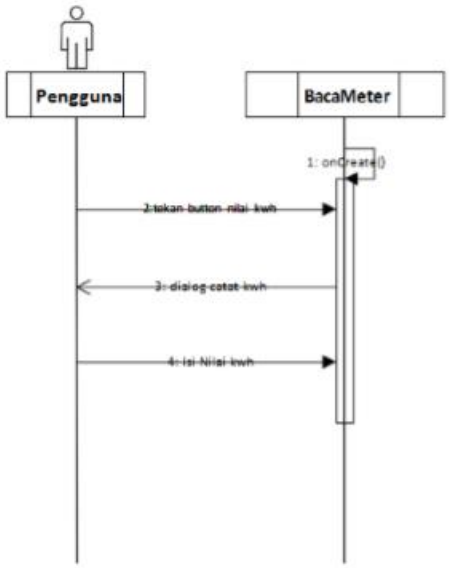
Gambar 2.4 Activity Diagram

2.8.5 Sequence Diagram

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 terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai *respons* dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeLine* vertikal.

Message digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, message akan dipetakan menjadi operasi/metoda dari class. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message. Berikut Contoh *sequence* diagram pada gambar 2.5.



Gambar 2.5 Sequence Diagram

UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM [16].

2.9 WEB API

Web *API* adalah sebuah alamat web (*end point*) yang dibuat untuk menangani beberapa pekerjaan sesuai permintaan yang diterima. Web *API* terkadang memiliki parameter sebagai data yang dibutuhkan agar dapat menampilkan hasil yang diinginkan. Pada beberapa kasus untuk mengakses *API* dibutuhkan kode otentikasi yang telah diizinkan untuk melihat data yang diinginkan. Semua rule ini ditentukan oleh programmer yang membuatnya [17]

2.10 Flask

Flask adalah web *framework* dari bahasa *python*. Dia menyediakan, *libraries* dan kumpulan kode yang bisa kamu gunakan untuk membangun *website*, tanpa perlu melakukan semuanya dari nol.

Karena fiturnya yang sederhana, flask akan lebih ringan dan tidak tergantung dengan banyak *library* luar yang perlu diperhatikan. Secara umum flask menyediakan '*Wekzeug*' yang berguna untuk menerima

request (*url*) dan memberikan respon dan ada Jinja2, *template engine* yang digunakan untuk menampilkan data dan menulis logika pada tampilannya

2.11 Heroku

Heroku adalah *platform cloud* sebagai layanan yang mendukung beberapa bahasa pemrograman. Salah satu *platform cloud* pertama, Heroku telah dikembangkan sejak Juni 2007, ketika itu hanya mendukung bahasa pemrograman Ruby, tetapi sekarang mendukung *Java*, *Node.js*, *Scala*, *Clojure*, *Python*, *PHP*, dan *Go*.