BAB 2
LANDASAN TEORI

2.1 Kecerdasan Buatan

Kecerdasan buatan sering disebut *Artificial Intelligence* (AI), kecerdasan *artificial*, *intelijensia artificial*, atau *intelijensia* buatan. Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudung pandang mereka masing-masing. Ada yang fokus pada logika berpikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokkan definisi AI yang diperoleh dari beberapa *textbook* berbeda ke dalam empat kategori [8], yaitu :

1. *Thinking Humanly*

   Pendekatan ini dilakukan dengan dua cara sebagai berikut :

   a. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berfikir.
   b. Melalui eksperimen-eksperimen psikologi.

2. *Acting Humanly*

   Pada tahun 1950, Alan Turing merancang suatu kajian bagi komputer berinteligensia untuk menguji apakah komputer tersebut mampu mengelabui seorang manusia yang menginterogasinya melalui telepati (komunikasi berbasis teks jarak jauh). Jika interrogator tidak dapat membedakan yang diinterogasi adalah manusia atau komputer, maka komputer berinteligensia tersebut lolos dari *Turing test*. Komputer tersebut perlu memiliki kemampuan *Natural Language Processing*, *Knowledge Representation, Automated Reasoning, Machine Learning, Computer Vision*, dan *Robotic*. Turing Test sengaja menghindari interaksi fisik antara interrogator dan komputer karena simulasi fisik manusia tidak memerlukan intelegensia.

3. *Thinking Rationally*

   Terdapat dua masalah dalam pendekatan ini, yaitu :

a. Tidak mudah untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut ke dalam formal term yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.
b. Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

4. Acting Rationally

Membuat inferensi yang logis merupakan bagian dari suatu *rational agent.* Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka agen dapat melakukan aksi berdasarkan kesimpulan tersebut.

2.2 Pengolahan Citra Digital

Pengolahan citra digital artinya melakukan pemrosesan gambar berdimensi dua melalui komputer digital. Menurut Efford (2000), pengolahan citra adalah istilah umum untuk berbagai teknik yang keberadaannya untuk memanipulasi dan memodifikasi citra dengan berbagai cara [9]. Foto adalah contoh gambar berdimensi dua yang dapat diolah dengan mudah. Setiap foto dalam bentuk citra digital dapat diolah menggunakan perangkat lunak tertentu. Misalnya citra yang gelap dapat diolah supaya menjadi lebih terang, mengubah format warna citra menjadi *grayscale*, *resize* citra dan lain sebagainya.

2.2.1 *Grayscale*

Citra *grayscale* menangani gradasi warna hitam dan putih, yang tentu saja menghasilkan efek warna abu-abu. Warna gambar dinyatakan dengan intensitas. Dalam hal ini, intensitas berkisar antara 0 sampai dengan 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih [9]. Berikut adalah rumus konversi citra berwarna (RGB) menjadi *grayscale* dapat dilihat pada rumus (2.1).

$$I = (0.2989 * R) + (0.5870 * G) + (0.1141 * B) \quad (2.1)$$

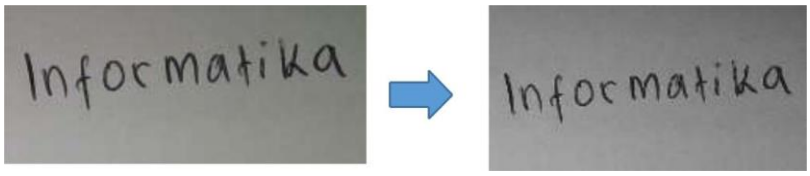
Keterangan :

R = komponen nilai merah (*Red*) dari suatu titik *pixel*

G = komponen nilai hijau (*Green*) dari suatu titik *pixel*

B = komponen nilai biru (*Blue*) dari suatu titik *pixel*

Persamaan di atas merupakan salah satu rumus yang digunakan untuk mengonversi citra berwarna menjadi *grayscale*. Persamaan 2.1 dipilih dalam penelitian ini karena mata manusia secara alami lebih sensitif terhadap cahaya merah dan hijau. Maka dari itu, warna-warna ini diberi bobot yang lebih tinggi untuk memastikan bahwa keseimbangan intensitas relatif dalam citra *grayscale* yang dihasilkan mirip dengan citra warna RGB [9]. *Grayscale* dapat dilihat pada Gambar 2.1.



Gambar 2. 1 Grayscale

2.2.2 *Sauvola Threshold*

Sauvola Threshold merupakan metode *threshold* yang mampu memberikan peningkatan dengan cara menghitung ambang menggunakan rentang nilai dinamis dari nilai standar deviasi citra *grayscale* [10]. *Sauvola* termasuk dalam *local threshold* di mana nilai ambang ditentukan oleh nilai *pixel* tetangga. Berikut ini adalah perhitungan nilai *pixel* dapat dilihat pada rumus (2.2).

$$\mathrm{T(x,y)} = m(x,y) * (1 + k * \left(\frac{s(x,y)}{R} - 1\right))) \qquad (2.2)$$

Pada rumus (2.2) terdapat rumus untuk menghitung m(*x*,*y*) yang dapat dihitung menggunakan rumus (2.3) dan rumus untuk menghitung s(*x*,*y*) dapat dilihat pada rumus (2.4).

$$\mathrm{m(x,y)} = \frac{\sum_{i=min}^{i=max} \sum_{j=min}^{j=max} img\,(i,j)}{i * j} \qquad (2.3)$$

$$s(x,y) = \sqrt{\frac{\sum_{i=min}^{i=max}\sum_{j=min}^{j=max} img(i,j) - m - (x,y))^2}{(i*j)-1}} \quad (2.4)$$

Keterangan :

R : Nilai maksimum dari standar deviasi (128 untuk citra *grayscale*)

k : *Kernel* dengan nilai antara 0.2 – 0.5.

m : Fungsi yang menghasilkan nilai rata-rata dari sejumlah *pixel* citra.

s : Fungsi yang menghasilkan nilai standar deviasi dari sejumlah *pixel* citra

T : Fungsi yang menghasilkan nilai *threshold* (ambang)

x : Nilai koordinat lebar citra

y : Nilai koordinat tinggi citra

Setelah nilai ambang T(x,y) sudah didapatkan, selanjutnya masukkan ke persamaan (2.5) di bawah ini.

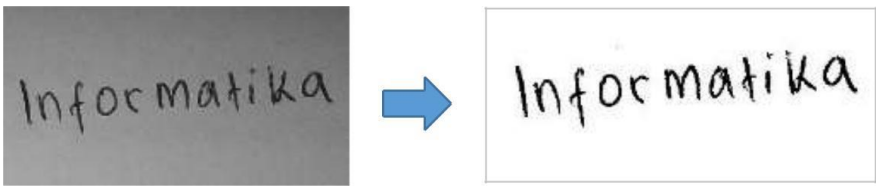
$$f(x,y) = \begin{cases} 0, & img(x,y) < T(x,y) \\ 255, & img(x,y) \geq T(x,y) \end{cases} \quad (2.5)$$

Keterangan :

f : Fungsi yang menghasilkan nilai 0 atau 255

img : Nilai *grayscale* citra

Hasil dari proses *sauvola threshod* dapat dilihat pada Gambar 2.2.



Gambar 2. 2 Sauvola Threshold

2.2.3 Operasi *Dilasi*

Operasi dilasi digunakan untuk memperbaiki citra yang pecah. Bila suatu objek (citra *input*) dinyatakan dengan A dan SE dinyatakan dengan B serta Bx menyatakan translasi B sedemikian sehingga pusat B terletak pada x. Operasi dilasi A dengan B dapat dinyatakan sebagai berikut []

$$D(A,B)=A\oplus B=\{x:B_x\cap A\neq\emptyset\}$$

Dengan $\emptyset$ menyatakan himpunan kosong.

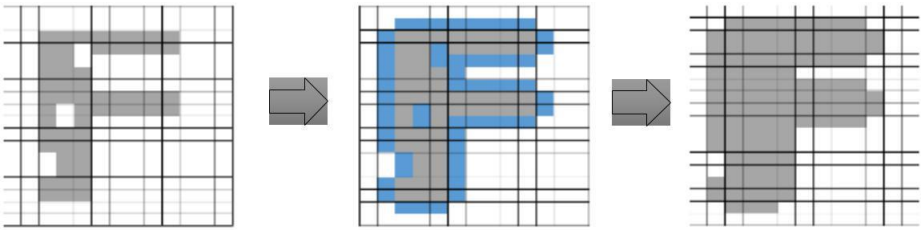
Keterangan :

D : Fungsi yang menghasilkan citra *dilasi*

A : Citra *input*

B : *Structuring Elements* (SE)

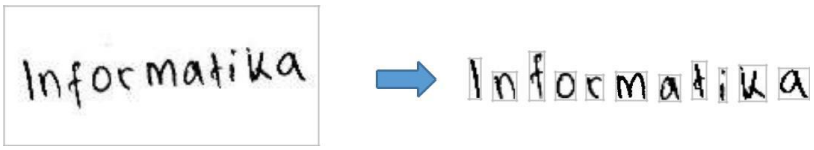
Proses *dilasi* dilakukan dengan membandingkan setiap *pixel* citra *input* dengan nilai pusat SE dengan cara (*superimpose*) SE dengan citra sehingga pusat SE tepat dengan posisi *pixel* citra yang di proses. Jika paling sedikit ada 1 pixel pada SE sama dengan nilai *pixel* objek (*foreground*) citra maka *pixel* input diset nilainya dengan nilai *pixel foreground* dan bila semua *pixel* yang berhubungan adalah *background* maka maka input *pixel* diberi nilai *background*. Proses serupa dilanjutkan dengan menggerakkan (translasi) SE *pixel* demi *pixel* pada citra input Berikut adalah contoh matriks citra *dilasi* dengan menggunakan *structure element* 3x3.



Gambar 2. 3 Contoh Dilasi

2.2.4 Segmentasi

Segmentasi merupakan teknik untuk membagi suatu citra menjadi beberapa daerah (*region*) di mana setiap daerah memiliki kemiripan atribut [11]. Sampai sejauh mana pembagian tersebut dilakukan tergantung pada masalah yang dihadapi. Idealnya, langkah segmentasi tersebut dihentikan pada saat objek yang diinginkan sudah berhasil dipisahkan. Dalam penelitian ini segmentasi dilakukan dengan memotong baris dan kolom citra. Pemotongan pertama dilakukan untuk semua baris, kemudian dilanjutkan dengan pemotongan kolom. Segmentasi dapat dilihat pada Gambar 2.4.



Gambar 2. 4 Segmentasi citra

2.2.5 *Resize*/Normalisasi

Resize citra artinya mengubah besarnya ukuran citra digital dalam *pixel*. Adakala ukurannya berubah menjadi lebih kecil dari ukuran aslinya dan ada kalanya sebaliknya. Proses *Resize* citra dapat mempengaruhi *size* citra lebih besar atau lebih kecil dan juga kualitas citra menjadi lebih baik atau lebih buruk. *Resize*/Normalisasi dapat dilihat pada Gambar 2.5



Gambar 2. 5 Resize/Normalisasi

Contoh *resize* di atas mengubah citra yang awalnya memiliki ukuran 35x32 *pixel* menjadi 20x40 *pixel*.

2.2.5 Thinning

Thinning adalah proses pengurangan data yang mengikis (erosi) sebuah objek hingga menjadi 1 ukuran *pixel* dan menghasilkan kerangka (*skeleton*) dari objek tersebut. Objek seperti huruf atau silhouettes dapat lebih mudah dikenali dengan melihat kepada kerangkanya saja [11]. Pada penelitian ini digunakan algoritma *thinning Zhang-Suen*. Algoritma ini adalah algoritma untuk citra biner, di mana piksel background citra bernilai 0, dan *pixe*l *foreground* (region) bernilai 1. Algoritma ini cocok digunakan untuk bentuk yang diperpanjang (elongated) dan dalam aplikasi OCR (*Optical Character Recognition*). Algoritma ini terdiri dari beberapa penelusuran, di mana setiap penelusurannya terdiri dari 2 langkah dasar yang diaplikasikan terhadap titik yang pikselnya bernilai 1, dan memiliki

paling sedikit 1 piksel dari 8-tetangganya yang bernilai 0. Gambar 2.3 mengilustrasikan titik objek P1 dan 8-tetangganya:

P9	P2	P3
P8	**P1**	P4
P7	P6	P5

Gambar 2. 6 Pixel P1 dengan 8 tetangga

Langkah pertama dari sebuah penelusuran adalah menandai semua titik objek untuk dihapus, jika titik objek tersebut memenuhi syarat-syarat berikut:

a. $2 \leq N(P1) \leq 6$
b. $S(P1) = 1$
c. $P2 * P4 * P6 = 0$
d. $P4 * P6 * P8 = 0$

Keterangan :

1. Jumlah dari tetangga titik objek P1, yang pikselnya bernilai 1, yaitu:
   $N(P1) = P2 + P3 + P4 + \ldots + P9$
2. S(P1) adalah jumlah perpindahan nilai nilai dari 0 (nol) ke 1 (satu) mulai dari P2, P3, …, P8, P9.
3. P2 * P4 * P6 = 0, memiliki arti P2 atau P4 atau P6 bernilai 0 (nol). Dan pada langkah kedua, kondisi (a) dan (b) sama dengan langkah pertama, sedangkan kondisi (c) dan (d) diubah menjadi:
   (c') P2 * P4 * P8 = 0
   (d') P2 * P6 * P8 = 0

2.3 Transformasi *Curvelet*

Dalam Transformasi *Cuverlet*, proses transformasi dilakukan secara linier dan mengambil masukan dari sebuah citra dalam bentuk *array cartesian* $[t_1,t_2]$, $0 \leq t_1,t_2 < n$, didapatkanlah keluaran sebagai *array* dari kumpulan koefisien $c^D(j,l,k)$ dari (l). Rumus untuk menghitung koefisien $c^D(j,l,k)$ dapat dilihat pada persamaan (2.6).

$$c^d(\mathrm{j,l,k}) = \sum_{0 \le t1,t2 < n} f[t1,t2]\, \phi^{k}_{j,l,k}\ [t1,t2] \qquad (2.6)$$

Di mana setiap $\phi^{k}_{j,l,k}\ [t1,t2]$ adalah digital curvelet waveform. DCT dapat diimplementasikan dengan dua cara. Metode pertama berdasar pada *Unequally-Spaced Fast Fourier transform* (USFFT) dan yang kedua berdasar pada Wrapping sampel Fourier dipilih secara khusus. Kedua implementasi pada dasarnya berbeda dengan *grid* spasial yang digunakan untuk menerjemahkan *curvelet* pada setiap skala dan sudut. Pada Tugas Akhir ini digunakan metode yang menggunakan *Wrapping*.

1. Aplikasikan FFT 2D untuk mengubah ke domain frekuensi dan dapatkan sample Fourier pada tiap skala.
2. Pada tiap skala dan sudut, resample atau interpolasi tiap sample fourier tadi untuk mendapatkan nilai sample pada setiap wedge.
3. Kalikan matriks pada wedge yang telah di interpolasi dengan window sinus/elips Curvelet.
4. Aplikasikan invers FFT 2D pada tiap wedge dan didapatkanlah koefesien Curvelet pada tiap wedge dan skala.

Pada tugas akhir ini, dari koefesien Curvelet tiap wedge ini dicari energi dan standar deviasi yang digunakan sebagai vektor fitur.

2.4 Teknik Learning (Belajar)

Teknik *learning* merupakan salah satu bagian dari ilmu kecerdasan buatan. Dalam *learning* kita tidak harus tahu aturan yang berlaku dalam sistem yang dibuat, melainkan aturan yang diharapkan bisa secara otomatis ditemukan. Proses belajar dalam teknik *learning* menggunakan data-data masukan sebagai pengalaman yang baru untuk dapat meningkatkan per formasinya dari sistem. Program komputer yang sanggup belajar adalah program yang bisa meningkatkan per formasinya melalui pengalaman [8]. Ada beberapa metode yang menggunakan teknik *learning* seperti *decision tree*, jaringan saraf tiruan, algoritma genetika dan lain-lain.

2.5 Jaringan Saraf Tiruan (JST)

Jaringan Saraf Tiruan (JST) adalah prosesor tersebar paralel (*parallel distributed processor*) yang sangat besar yang memiliki kecenderungan untuk menyimpan pengetahuan yang bersifat pengalaman dan membuatnya siap untuk digunakan [8]. JST menyerupai otak manusia dalam dua hal, yaitu :

1. Pengetahuan diperoleh jaringan melalui proses belajar
2. Kekuatan hubungan antar sel saraf (*neuron*) yang dikenal sebagai bobot-bobot sinaptik digunakan untuk menyimpan pengetahuan.

JST merupakan salah satu upaya manusia untuk memodelkan cara kerja atau fungsi sistem saraf manusia dalam melaksanakan tugas tertentu. Permodelan ini didasari oleh kemampuan otak manusia dalam mengorganisasikan sel-sel penyusunnya yang disebut *neuron*, sehingga mampu melaksanakan tugas-tugas tertentu, khususnya pengenalan pola dengan efektivitas yang sangat tinggi.

2.5.1 Model Sel Saraf (*Neuron*)

Sel saraf (*neuron*) adalah unit pemrosesan informasi yang merupakan dasar dari operasi JST. Terdapat tiga elemen dasar dari model *neuron* [8], yaitu :

1. Sekumpulan sinapsis atau jalur hubungan, di mana masing-masing sinapsis memiliki bobot atau kekuatan hubungan.
2. Suatu *adder* (instruksi) untuk menjumlahkan sinyal-sinyal *input* yang diberi bobot oleh sinapsis *neuron* yang sesuai. Operasi-operasi yang digambarkan di sini mengikuti aturan *linear combiner*.
3. Suatu fungsi aktivasi untuk membatasi amplitudo *output* dari setiap *neuron*.

2.5.2 Arsitektur Jaringan

Arsitektur jaringan merupakan pola di mana *neuron-neuron* pada JST disusun berhubungan erat dengan algoritma belajar yang digunakan untuk melatih jaringan. Secara umum arsitektur jaringan dibagai menjadi empat [8], yaitu :

1. Single-Layer Feedforward Networks

Suatu JST berlapis adalah jaringan *neuron* yang diorganisasikan dalam bentuk lapisan-lapisan. Pada bentuk jaringan berlapis yang paling sederhana, hanya terdapat *input layer* dengan *node* sumber yang terproyeksi ke dalam *output*

layer dari *neuron*, tetapi tidak sebaliknya. Jaringan ini adalah jaringan jenis *feedforward* yang tepat.

2. *Multi-Layer Feedforward Networks*

   Kelas ke dua dari *feedforward neural network* adalah jaringan dengan satu atau lebih lapis tersembunyi (*hidden layer*), dengan c*omputation nodes* yang berhubungan disebut *hidden neurons* atau *hidden units.*

3. *Recurrent Networks*

   Recurrent Neural Network adalah jaringan yang mempunyai minimal satu *feedback loop.* Suatu *recurrent network* bisa terdiri dari satu lapisan *neuron* tunggal dengan masing-masing *neuron* memberikan kembali *outputnya* sebagai *input* pada semua *neuron* yang lain.

4. *Lattice Structure*

   Sebuah *lattice* terdiri dari satu dimensi, dua dimensi, atau lebih *array neuron* dengan himpunan *node* sumber yang bersesuaian dan memberikan sinyal *input* ke *array*. Dimensi *lattice* pada jumlah dimensi ruang di mana *graph* berada.

Dalam penelitian ini, arsitektur jaringan yang digunakan adalah *SingleLayer Feedforward Networks* di mana nilai masukan yang masuk pada jaringan langsung diolah tanpa melalui *hidden layer* dan kemudian setelah itu masuk ke lapisan *output* untuk mendapatkan hasilnya.

2.5.3 Proses Belajar JST

Belajar dalam konteks JST adalah proses di mana parameter-parameter bebas JST diadaptasikan melalui suatu proses perangsangan berkelanjutan oleh lingkungan di mana jaringan berada. Proses belajar dalam JST ada dua [8], yaitu :

1. *Supervised Learning* (Belajar Dengan Pengawasan)

   Supervised atau *active learning* adalah proses belajar membutuhkan guru. Yang dimaksud guru di sini adalah sesuatu yang memiliki pengetahuan tentang lingkungan. Guru bisa direpresentasikan sebagai sekumpulan sampel *input-output*. Pembangunan pengetahuan dilakukan oleh guru dengan memberikan respons yang diinginkan kepada JST. Parameter-parameter jaringan berubah-ubah berdasarkan vektor latih dan sinyal kesalahan (sinyal kesalahan adalah perbedaan antara keluaran JST dan respons yang diinginkan). Proses perubahan

ini dilakukan secara berulang-ulang, selangkah demi selangkah, dengan tujuan agar JST bisa memiliki kemampuan yang mirip dengan gurunya. Dengan kata lain, JST dilatih untuk dapat memetakan sekumpulan sampel *input-output* dengan akurasi yang tinggi. Pada proses belajar ini, output yang diharapkan telah diketahui sebelumnya.

2. *Unsupervised Learning* (Belajar Tanpa Pengawasan)

   Unsupervised atau *self-organized learning* tidak membutuhkan guru untuk memantau proses belajar. Dengan kata lain, tidak ada sekumpulan sampel *input-output* atau fungsi tertentu untuk dipelajari oleh jaringan. Salah satu contoh *unsupervised learning* adalah *competitive learning*. Pada metode ini, tidak ditentukan hasil yang seperti apakah yang diharapkan selama proses pembelajaran.

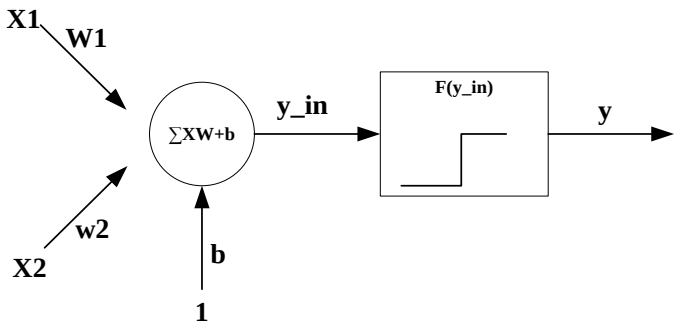
Proses belajar yang digunakan dalam penelitian ini yaitu *supervised learning* (pembelajaran terawasi). *Supervised learning* digunakan karena target kelas yang didapatkan sudah ditentukan sebelumnya.

2.5.4 Pembelajaran Dengan Pengawasan (*Supervised Learning*)

Ada banyak metode pembelajaran yang termasuk dalam *supervised learning* dan sudah pernah diimplementasikan dalam beberapa kasus. Namun dalam penelitian ini, metode pembelajaran yang digunakan adalah metode *Backpropagation*. Di bawah ini dijelaskan beberapa metode pembelajaran yang termasuk ke dalam *supervised learning*.

1. *Hebb Rule*

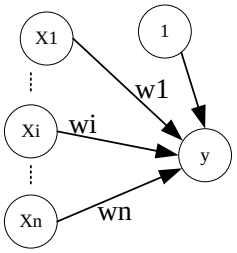
   Dalam metode *hebb rule* proses pembelajaran terjadi dengan cara memodifikasi dari bobot *sinapsis* atau bobot *sinaptic*, jika dua neuron yang aling berhubungan satu dengan yang lain dalam waktu dan kondisi yang sama, maka bobot dari kedua neuron tersebut akan dinaikkan. *Hebb rule* dapat dilihat pada Gambar 2.7



Gambar 2. 7 Hebb Rule

2. Perceptron

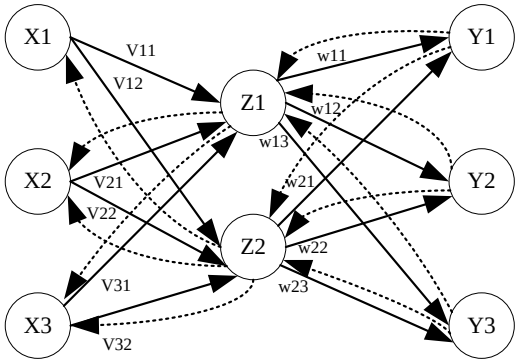
Metode *perceptron* merupakan suatu metode pelatihan yang lebih kuat dari metode *hebb*, terutama dalam iterasi dapat membuat *output* dari bobot menjadi konvergen. *Perceptron* dapat dilihat pada Gambar 2.8.



Gambar 2. 8 Perceptron

3. Backpropagation

Pelatihan *Backpropagation* meliputi tiga fase. Fase pertama adalah fase maju. Pola masukan dihitung maju mulai dari layar masukan hingga layar keluaran menggunakan fungsi aktivasi yang ditentukan. Fase kedua adalah fase mundur. Selisih antara keluaran jaringan dengan target yang diinginkan merupakan kesalahan yang terjadi. Kesalahan tersebut dipropagasikan mundur, dimulai dari garis yang berhubungan langsung dengan unit-unit di layar keluaran. Fase ketiga adalah modifikasi bobot untuk menurunkan kesalahan yang terjadi [8]. Berikut adalah arsitektur dari jaringan *backpropagation* pada Gambar 2.9.



Gambar 2. 9 Backpropagation

a. Fase I : Propagasi maju

Selama propagasi maju, sinyal masukan (=xi) dipropagasikan ke layar tersembunyi menggunakan fungsi aktivasi yang ditentukan. Keluaran dari setiap unit layar tersembunyi (=zj) tersebut selanjutnya dipropagasikan maju lagi ke layar tersembunyi di atasnya menggunakan fungsi aktivasi yang ditentukan. Demikian seterusnya hingga menghasilkan keluaran jaringan (=yk).

Berikutnya, keluaran jaringan (=yk) dibandingkan dengan target yang harus dicapai (=tk). Selisih tk - yk adalah kesalahan yang terjadi. Jika kesalahan ini lebih kecil dari batas toleransi yang ditentukan, maka iterasi dihentikan. Akan tetapi apabila kesalahan masih lebih besar dari batas toleransinya, maka bobot setiap garis dalam jaringan akan dimodifikasi untuk mengurangi kesalahan yang terjadi.

b. Fase II : Propagasi mundur

Berdasarkan kesalahan tk - yk, dihitung faktor δk (k = 1,2 , ... , m) yang dipakai untuk mendistribusikan kesalahan di unit yk ke semua unit tersembunyi yang terhubung langsung dengan yk. δk juga dipakai untuk mengubah bobot garis yang berhubungan langsung dengan unit keluaran.

Dengan cara yang sama, dihitung faktor δj di setiap unit di layar tersembunyi sebagai dasar perubahan bobot semua garis yang berasal dari unit tersembunyi di layar di bawahnya. Demikian seterusnya hingga semua faktor δ di unit tersembunyi yang berhubungan langsung dengan unit masukan dihitung.

c. Fase III : Perubahan bobot

Setelah semua faktor δ dihitung, bobot semua garis dimodifikasi bersamaan. Perubahan bobot suatu garis didasarkan atas faktor δ neuron di layar atasnya. Sebagai contoh, perubahan bobot garis yang menuju ke layar keluaran didasarkan atas δk yang ada di unit keluaran.

Ketiga fase tersebut diulang-ulang terus hingga kondisi penghentian dipenuhi. Umumnya kondisi penghentian yang sering dipakai adalah jumlah iterasi atau kesalahan. Iterasi akan dihentikan jika jumlah iterasi yang dilakukan sudah melebihi jumlah maksimum iterasi yang ditetapkan, atau jika kesalahan yang terjadi sudah lebih kecil dari batas toleransi yang diizinkan.

2.6 Algoritma *Rabin Karp*

Algoritma *Rabin-karp* adalah algoritma pencocokan string yang menggunakan fungsi *hash* sebagai pembanding antara *string* yang dicari (m) dengan *substring* pada teks (n). Algoritma *Rabin-karp* didasarkan pada fakta jika dua buah *string* sama maka harga *hash value*-nya pasti sama. Akan tetapi ada dua masalah yang timbul dari hal ini, masalah pertama yaitu ada begitu banyak *string* yang berbeda, permasalahan ini dapat dipecahkan dengan meng-*assign* beberapa *string* dengan *hash value* yang sama. Masalah yang kedua belum tentu *string* yang mempunyai *hash value* yang sama cocok untuk mengatasinya maka untuk setiap *string* yang di-*assign* dilakukan pencocokan *string* secara Brute-Force. Kunci agar algoritma Rabin-Karp efisien, terdapat pada pemilihan *hash value*-nya. Salah satu cara yang efektif adalah memperlakukan setiap *substring* sebagai suatu bilangan dengan basis tertentu [3]. Tahap-tahap yang dilakukan dalam evaluasi jawaban esai yaitu

proses *tokenizing* dan *case folding*, *filtering*, algoritma Rabin-Karp (*Kgram*, *hashing*, *similarity*).

2.6.1 *Tokenizing* dan *Case Folding*

Pada tahan *tokenizing* dan *case folding* dilakukan proses pemisahan teks menjadi per kata, kemudian mengubah semua kata yang sudah ditoken menjadi huruf kecil (*lowercase*). Berikut adalah contoh *Tokenizing* dan *Case Folding* dapat dilihat pada Tabel 2.1.

Tabel 2. 1 Contoh *Tokenizing* dan *Case Folding*

Teks	Tokenizing dan Case Folding
Bandung ada di Provinsi Jawa Barat	Bandung ada di provinsi jawa barat

2.6.2 *Filtering*

Filtering adalah tahap mengambil kata-kata penting dari hasil *tokenizing* dan *case folding*, dengan menggunakan *stoplist* / *stopword* (membuang kata yang kurang penting). Contoh kata yang termasuk ke dalam *stoplist* seperti : ‘di’, ‘yaitu’, ‘adalah’, ‘merupakan’, ‘dan’, ‘yang’, ‘ini’. Berikut adalah contoh dari proses *filtering*. Berikut adalah contoh dari proses *filtering* dapat dilihat pada Tabel 2.2.

Tabel 2. 2 Contoh Proses *Filtering*

Proses	Hasil
Hasil *tokenizing* dan *case* folding	Bandung Ada Di Provinsi Jawa

	barat
Hasil proses filtering	Bandung Provinsi Jawa barat

Pada contoh di atas kata ‘ada’ dan ‘di’ dihilangkan karena kata tersebut termasuk dalam *stoplist* / *stopword*, sehingga kata yang akan diproses dalam algoritma *Rabin-Karp* dapat dilihat pada Tabel 2.3.

Tabel 2. 3 Contoh Algoritma Rabin-Karp

Teks	Hasil proses filtering
Bandung Ada Di Provinsi Jawa barat	Bandung, provinsi, jawa, barat

2.6.3 K-gram

K-gram adalah rangkaian *terms* dengan panjang K. Kebanyakan yang digunakan sebagai *terms* adalah kata. K-gram merupakan sebuah metode yang diaplikasikan untuk pembangkitan kata atau karakter. Metode K-gram ini digunakan untuk mengambil potongan-potongan karakter huruf sejumlah k dari sebuah kata yang secara kontinuitas dibaca dari teks sumber hingga akhir dari kalimat atau dokumen [3]. Berikut adalah contoh K-gram dengan nilai k=4.

Tabel 2. 4 Contoh K-gram

Teks	Hasil proses K-gram
Bandungprovinsijawabarat	Band, andu, ndun, dung, ungp, ngpr, gpro, prov, rovi, ovin, vins, insi, nsij, sija, ijaw, jawa, awab, waba, abar, bara, arat

2.6.4 Hashing

Hashing adalah suatu cara untuk mentransformasi sebuah *string* menjadi suatu nilai yang unik dengan panjang tertentu (*fixed-length*) yang berfungsi sebagai penanda *string* tersebut. Fungsi untuk menghasilkan nilai ini disebut fungsi *hash*, sedangkan nilai yang dihasilkan disebut nilai *hash*. Nilai *hash* pada umumnya digambarkan sebagai *fingerprint* yaitu suatu *string* pendek yang terdiri atas huruf dan angka yang terlihat acak (data biner yang ditulis dalam heksadesimal) [3]. Berikut ini adalah untuk mendapatkan nilai *hasing* untuk setiap K-gram dapat dilihat pada persamaan (2.7).

$$H = C_1 * a^{(k-1)} + C_2 * a^{(k-2)} + C_3 * a^{(k-3)} \ldots C_k * a^{()} \quad (2.7)$$

Keterangan :

H : Nilai *hashing*

Ck : Nilai ascii dari setiap huruf pada K-grams

a : nilai konstanta / basis

k : Banyak karakter

2.6.5 Similarity

Similarity digunakan untuk menghitung tingkat kemiripan suatu teks, di mana yang dibandingkan adalah nilai hasil proses *hashing*. Untuk mengukur persentase kemiripan menggunakan metode *Dice's Similarity Coefisient* di mana *hash value* yang sama dari dua dokumen teks abstrak yang berbeda dihitung dan jumlahnya disimpan dalam bentuk array. Berikut adalah persamaan *similarity* metode *Dice's Similarity Coefisient.*

$$s = \frac{2*C}{A*B} \quad (2.8)$$

Keterangan :

S = Nilai *similarity* antar kedua teks

A = Jumlah kumpulan K-grams dalam teks 1

B = Jumlah kumpulan K-grams dalam teks 2

C = Jumlah dari K-grams yang sama dari teks yang dibandingkan

Penggunaan *similarity* dalam penelitian ini untuk melihat tingkat kemiripan antara teks kunci jawaban dengan teks hasil pengenalan tulisan tangan jawaban esai.

2.7 Unified Modeling Language (UML)

Unified Modeling Language (UML) adalah bahasa grafis untuk mendokumentasi, menspesifikasikan, dan membangun sistem perangkat lunak [25]. UML berorientasi objek, menerapkan banyak level abstraksi, tidak bergantung proses pengembangan tidak bergantung bahasa dan teknologi, pemaduan beberapa notasi di beragam metodologi, usaha bersama dari banyak pihak, didukung oleh kakas-kakas yang diintegrasikan lewat XML. Standar UML dikelola oleh OMG (*Object Management Group*). UML bukanlah :

1. Bahasa pemrograman visual, tetapi bahasa pemodelan visual.
2. Spesifikasi Kakas, tetapi spesifikasi bahasa pemodela.
3. Proses, Tetapi yang memungkinkan proses-proses.

Teknik pemodelan dalam UML ada dua jenis diagramnya [12], yaitu :

1. Diagram struktur

   Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek statik dari sistem. Diagram struktur UML terdiri dari : Diagram kelas (*Class diagram*), Diagram objek (*Object diagram*), Diagram Komponen (*Component diagram*), dan Diagram deployment (*Deployment diagram*).
2. Diagram prilaku

   Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek dinamis dari sistem. Diagram prilaku dalam UML terdiri dari : Diagram use case (*Use case diagram*), Diagram sekuen (*Sequence diagram*), Diagram kolaborasi (*Collaboration diagram*), Diagram aktivitas (*Activity Diagram*).

2.7.1 Use Case Diagram

Use Case diagram merupakan salah satu diagram untuk memodelkan aspek tingkah laku sistem. Masing-masing use case diagram menunjukkan

sekumpulan use case, aktor dan hubungannya. Use case diagram penting untuk memvisualisasikan, menspesifikasikan, dan mendokumentasikan kebutuhan perilaku sistem. Diagram-diagram use case merupakan pusat pemodelan prilaku sistem, subsistem dan kelas [12]. Use case adalah interaksi antara aktor eksternal dan sistem, hasil yang dapat diamati oleh aktor, berorientasi pada tujuan, dideskripsikan di use case diagram dan teks. Relasi anta use case dengan use case lainnya ada tiga jenis, yaitu

1. Relasi *Include*

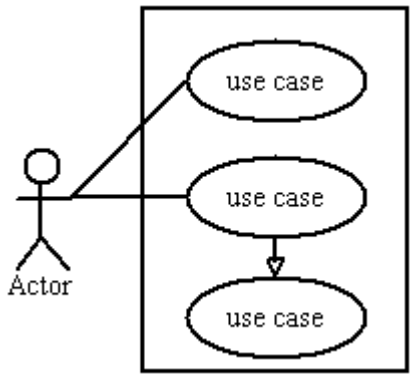
   Merupakan relasi antar use case di mana jika use case A meng-include use case B, maka use case B akan diimplementasikan setiap kali use case A diimplementasikan. Direpresentasikan dengan garis putus-putus bertuliskan <<include>> ke arah use case yang akan di-include.

2. Relasi *Extend*

   Merupakan relasi antar use case di mana jika use case A di extend oleh use case B maka use case B akan bisa saja diimplementasikan atau tidak setiap kali use case A diimplementasikan. Direpresentasikan dengan garis putusputus bertuliskan <<extend>> ke arah use case yang akan di-extend.

3. Relasi *Genealization*

   Digunakan untuk membuat use case lebih spesifik. Relasi *Genealization* dapat dilihat pada Gambar 2.9



Gambar 2. 10 Contoh Use Case Diagram

2.7.2 *Use Case Scenario*

Use case scenario merupakan hasil instansiasi dan penjelasan dari setiap use case. Use case scenario terbagi menjadi tiga bagian, yaitu:

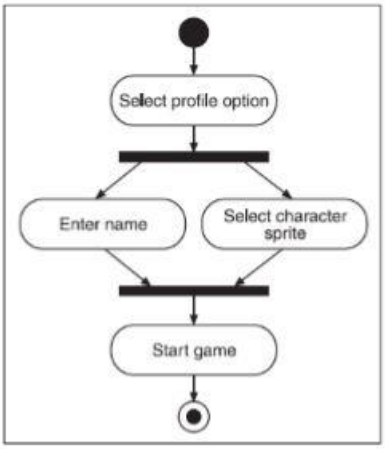
1. Identifikasi dan inisiasi
2. Step performed
3. Kondisi, asumsi dan pertanyaan.

2.7.3 *Activity* Diagram

Activity diagram adalah diagram *flowchart* yang diperluas yang menunjukkan aliran kendali satu aktivitas ke aktivitas lain. Kita menggunakan diagram ini memodelkan aspek dinamis sistem. Aktivitas adalah eksekusi notatomik yang berlangsung di *state machine*. *Activity diagram* mendeskripsikan aksi-aksi dan hasilnya. *Activity diagram* berupa operasi-operasi dan aktivitas-aktivitas di *use case*. *Activity diagram* dapat digunakan untuk :

1. Pandangan dalam yang dilakukan di operasi.
2. Pandangan dalam bagaimana objek-objek bekerja.
3. Pandangan dalam di aksi-aksi dan pengaruhnya pada objek-objek
4. Pandangan dalam dari suatu use case.
5. Logik dari proses bisnis.

Diagram aktivitas merupakan jenis khusus dan diagram *statechart*. *State* adalah aksi-aksi yang menuju state berikutnya setelah selesainya aksi itu. Contoh activity diagram dapat dilihat pada Gambar 2.10



Gambar 2. 11 Contoh Activity Diagram

Dari gambar 2.10 di atas merupakan contoh *activity diagram* dengan alur :

1. Lingkaran hitam penuh menandakan memulai aktivitas
2. Lingkaran hitam penuh menandakan memulai aktivitas
3. Masuk ke proses fork untuk menjalankan aktivitas secara pararel kemudian menjalankan aktivitas di bawahnya
4. Masuk ke proses join untuk menggabungkan proses yang dipisahkan oleh fork
5. Melakukan aktivitas Start game
6. Lingkaran hitam sebagian menandakan akhir dari suatu aktivitas

2.7.4 Class Diagram

Class diagram merupakan diagram paling umum dipakai di semua pemodelan berorientasi objek. Pemodelan kelas merupakan pemodelan yang paling utama di pendekatan berorientasi objek. Pemodelan kelas menunjukkan kelas-kelas yang ada di sistem dan hubungan antar kelas-kelas itu, atribut-atribut dan operasi - operasi di setiap kelas [12]. Class diagram menunjuk aspek statis sistem terutama untuk mendukung kebutuhan fungsional sistem. Meskipun class diagram serupa dengan model data, namun kelas-kelas tidak hanya menunjuk struktur informasi tapi juga mendeskripsikan perilaku. Salah satu maksud class diagram adalah untuk mendefinisikan fondasi bagi diagram-diagram lain di mana aspek-aspek lain dari sistem ditunjukkan. Elemen-elemen yang penting dalam class diagram adalah sebagai berikut [12].

1. Kelas

   Kelas merupakan elemen terpenting di sistem berorientasi objek. Kelas mendeskripsikan satu blok pembangunan sistem. Berikut adalah bagian dari kelas :

   a. Nama

      Nama kelas harus unik karena akan menjadi identifier di program.

   b. Atribut

      Atribut adalah properti bernama di kelas yang mendeskripsikan range nilai yang dipunyai instan kelas. Kelas dapat mempunyai sejumlah atribut atau tidak sama sekali.

c. Operasi

Operasi adalah implementasi layanan yang dapat diminta pada sembarang objek kelas itu untuk mempengaruhi prilaku sistem.

d. Acces Modifer

Access Modifier dalam kelas digunakan untuk membatasi akses dari kelas lain yang ingin mengakses atribut atau operasi dari suatu kelas. Ada tiga *Access Modifier* yang digunakan yaitu *public* (+), *protected* (#) dan *private* (-).

2. Antarmuka

Antarmuka (Interface) merupakan korelasi operasi yang menspesifisikan layanan dari suatu kelas atau komponen. Antarmuka mendeskripsikan perilaku tampak secara eksternal dari elemen.

3. Kolaborasi

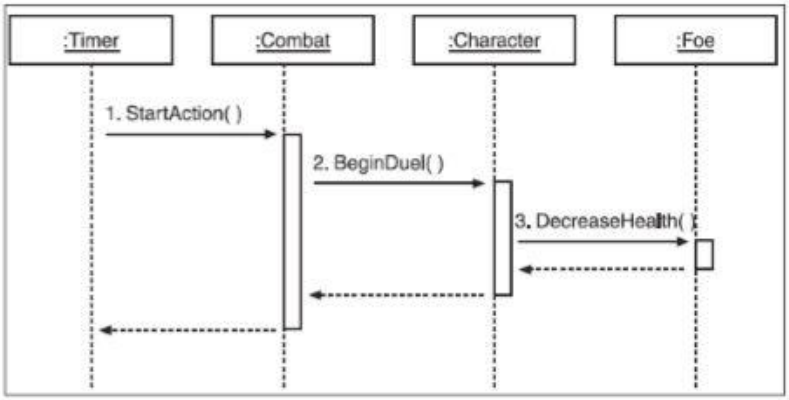
Kolaborasi merupakan pendefinisian suatu interaksi dan sekelompok peran elemen-elemen lain yang bekerja sama untuk menyediakan suatu perilaku kooperatif yang lebih besar dari penjumlahan seluruh elemen. Kolaborasi memiliki struktur, prilaku dan dimensi. Kolaborasi ini merepresentasikan implementasi pola tertentu yang membentuk sistem.

4. Hubungan

Hubungan antar kelas di diagram kelas terdiri dari Asosiasi dan Generalisasi.

2.7.5 Sequence Diagram

Sequence diagram digunakan untuk memodelkan skenario penggunaan. Skenario penggunaan adalah barisan kejadian yang terjadi selama satu eksekusi sistem. Cakupan skenario dapat beragam, dari mulai semua kejadian di sistem atau hanya kejadian pada objek-objek tertentu. Sequence diagram menunjukkan objek sebagai garis vertikal dan tiap kejadian sebagai panah horizontal dari objek pengirim ke objek pengirim ke objek penerima. Gambar 2.11 merupakan contoh *sequence diagram*.



Gambar 2. 12 Contoh Sequence Diagram

Dari contoh sequence diagram diatas memiliki empat kelas, di mana :

1. Kelas Timer memanggil fungsi StartAction() yang ada pada kelas Combat
2. Kelas Combat memanggil fungsi BeginDuel() yang terdapat pada kelas Character
3. Kelas Character memanggil fungsi DecreaseHealth() yang terdapat pada kelas Foe
4. Dari hasil pemanggilan fungsi, kemudian nilainya direturnkan kembali sampai ke kelas Timer

2.8 Pengujian Sistem

Pengujian adalah proses pemeriksaan atau evaluasi sistem atau komponen sistem secara manual atau otomatis untuk memverikasi apakah sistem memenuhi kebutuhan-kebutuhan yang dispesifikan atau mengidentifikasi perbedaan perbedaan antara hasil yang diharapkan dengan hasil yang terjadi [12]. Pengujian seharusnya meliputi tiga konsep berikut.

1. Demonstrasi validitas perangkat lunak pada masing-masing tahap di siklus pengembangan sistem
2. Penentuan validitas sistem akhir dikaitkan dengan kebutuhan pemakai
3. Pemeriksaan perilaku sistem dengan mengeksekusi sistem pada data sampel pengujian

Awalnya pengujian diartikan sebagai aktivitas yang dapat atau hanya dilakukan setelah pengkodean (kode program selesai). Namun, pengujian

seharusnya dilakukan dalam skala lebih luas. Pengujian dapat dilakukan begitu spesifikasi kebutuhan telah dapat didefinisikan. Evaluasi terhadap spesifikasi dan perancangan juga merupakan teknik di pengujian. Kategori pengujian dapat dikategorikan menjadi dua [12], yaitu :

1. Berdasarkan ketersediaan logik sistem, terdiri dari Black box testing dan White box testing
2. Berdasarkan arah pengujian, terdiri dari Pengujian top down dan Pengujian bottom up

4.8.1 Pengujian Black Box

Konsep *black box* digunakan untuk merepresentasikan sistem yang cara kerja di dalamnya tidak tersedia untuk diinspeksi. Di dalam *black box*, item-item yang diuji dianggap "gelap" karena logiknya tidak diketahui, yang diketahui hanya apa yang masuk dan apa yang keluar dari *black box* [12].

Pada pengujian *black box*, kasus-kasus pengujian berdasarkan pada spesifikasi sistem. Rencana pengujian dapat dimulai sedini mungkin di proses pengembangan perangkat lunak. Teknik pengujian konvensional yang termasuk pengujian "*black box*" adalah sebagai berikut [12].

1. Graph-based testing
2. Equivalence partitioning
3. Comparison testing
4. Orthogonal array testing

Pada pengujian *black box*, kita mencoba beragam masukan dan memeriksa keluaran yang dihasilkan. Kita dapat mempelajari apa yang dilakukan kotak, tapi tidak mengetahui sama sekali mengenai cara konversi dilakukan. Teknik pengujian *black box* juga dapat digunakan untuk pengujian berbasis skenario, di mana isi dalam sistem mungkin tidak tersedia untuk diinspeksi tapi masukan dan keluaran yang didefinisikan dengan *use case* dan informasi analisis yang lain.

4.8.2 Pengujian Akurasi

Akurasi merupakan seberapa dekat suatu angka hasil pengukuran terhadap angka sebenarnya (*true value* atau *reference value*). Tingkat akurasi diperoleh dengan persamaan (2.9) sebagai berikut.

$$Akurasi = \frac{Jumlah\ Karakter\ Sama}{Jumlah\ Seluruh\ Karakter} X100\% \quad (2.9)$$

2.9 Bahasa Pemrograman

Bahasa pemrograman diperlukan untuk menjalankan instruksi-instruksi apa yang harus dilakukan komputer. Komputer tidak bisa memahami bahasa manusia, sehingga diperlukan penggunaan bahasa komputer di dalam program komputer. Penelitian ini menggunakan bahasa pemrograman Java untuk mengembangkan aplikasi dan habasa pemrograman SQL untuk mengelola *database*.

2.9.1 C#

C# adalah bahasa pemrograman baru yang diciptakan oleh Microsoft yang dikembangkan di bawah kepemimpinan Anders Hejlsberg yang telah menciptakan berbagai macam bahasa pemrograman termasuk *Borland Turbo* C++ dan *orland Delphi*. Bahasa C# juga telah di standardisasi secara internasional oleh ECMA. Seperti halnya bahasa pemrograman yang lain, C# bisa digunakan untuk membangun berbagai macam jenis aplikasi, seperti aplikasi berbasis *windows* (desktop) dan aplikasi berbasis web serta aplikasi berbasis *web services*.

2.9.2 Structured Query Language

Structured Query Language (SQL) merupakan sebuah bahasa permintaan *database* yang terstruktur yang dibuat sebagai bahasa yang dapat menghubungkan beberapa tabel dalam *database* maupun menghubungkan antar *database*. Bahasa SQL ditulis langsung dalam sebuah program *database* sehingga pengguna dapat melihat langsung permintaan yang diinginkan dan sekaligus melihat hasilnya. Ada beberapa program *database* server yang menggunakan standar *query* berupa SQL yaitu : Oracle, *PostgreSQL, MySQL, MsQL, SQL Server*, dan lain-lain. Dalam penelitian ini SQL diimplementasikan pada *database* server *MySQL*.

2.10 Aplikasi

Aplikasi merupakan perangkat lunak siap pakai yang dapat digunakan untuk menjalankan perintah-perintah dari para pengguna aplikasi tersebut. Dalam penelitian ini digunakan aplikasi untuk mendukung penelitian yang dilakukan meliputi *Xampp* dan Visual Studio.

2.10.1 Xampp

Xampp merupakan *software* web server *apache* yang di dalamnya sudah tersedia *database* server *mysql* dan *support php programming*. *Xampp* juga mudah digunakan, gratis dan mendukung instalasi *Linux* dan Windows. Salah satu keuntungan *xampp* yaitu instalasi yang dilakukan satu kali dan sudah tersedia *Apache* Web Server, *MySql Database* Server, PHP *Support*, *FileZilla*, dan modul lainnya. Versi *Xampp* yang digunakan dalam penelitian ini adalah versi 5.6.19-0. Kemudian fitur yang digunakan adalah fitur *MySQL* dan *Apache* Web Server

2.10.2. Visual Studio

Microsoft Visual Studio merupakan sebuah perangkat lunak lengkap (suite) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi Windows, ataupun aplikasi Web. Visual Studio mencakup *compiler*, SDK, *Integrated Development Environment* (IDE), dan dokumentasi (umumnya berupa MSDN Library). *compiler* yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visual C#,Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe.

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code*(dalam bentuk Microsoft *Intermediate Language* di atas *.NET Framework*). Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi *Silverlight*, aplikasi Windows Mobile (yang berjalan di atas *.NET Compact Framework*).