BAB II
LANDASAN TEORI

2.1 Perbandingan dengan Penelitian Terdahulu

Ada beberapa penelitian terdahulu yang dijadikan penulis sebagai referensi. Antara lain adalah penelitian yang berkaitan dengan prediksi *Drought Code* (DC) dan algoritma GRLVQ. Untuk penjelasan lebih lanjut dapat dilihat pada Tabel 2.1.

Tabel 2.1 Perbandingan Dengan Penelitian Terdahulu

No	Peneliti	Judul	Persamaan	Perbedaan
1.	Midyanti, 2020	Kombinasi SOM-RBF Untuk Prediksi *Drought Code* Berdasarkan Data Curah Hujan dan Suhu Udara	- Masukkan yang diberikan berupa suhu udara dan curah hujan. - Keluaran berupa hasil prediksi *Drought Code*	- Data yang digunakan adalah data selama tiga hari. - Metode yang digunakan adalah kombinasi SOM dan RBF.
2.	Sinulingga, dkk, 2016	Pengenalan Wajah Menggunakan *Two Dimensional linear Discriminant Analysis* Berbasis Optimasi *Feature Fusion Strategy*	Metode yang digunakan adalah GRLVQ	GRLVQ digunakan untuk mengklasifikasi citra wajah.
3.	Hameed, dkk, 2021	*Application of Generalized Relevance Linear Vector Quantization for Diabetes Diagnosis*	Metode yang digunakan adalah GRLVQ	GRLVQ digunakan untuk mengklasifikasi penyakit diabetes.

2.2 Prediksi

Pengertian prediksi sama dengan ramalan atau perkiraan. Berdasarkan kamus besar bahasa Indonesia, prediksi merupakan hasil dari kegiatan meramal atau memperkirakan nilai pada masa yang akan datang dengan menggunakan data masa lalu. Prediksi menunjukkan apa yang akan terjadi pada suatu keadaan tertentu. Prediksi bisa berdasarkan metode ilmiah ataupun subjektif belaka. Prediksi tidak memberikan jawaban pasti, namun mencari jawaban sedekat mungkin dengan yang akan terjadi (Kafil, 2019).

2.3 *Automatic Weather Station* (AWS)

Automatic Weather Station adalah sebuah sistem yang didesain untuk memantau cuaca serta mengumpulkan dan memproses data secara otomatis. Data hasil pengukuran sistem AWS diproses secara lokal di lokasi AWS itu sendiri atau dikumpulkan pada suatu pusat data akusisi. Selanjutnya, secara otomatis data diteruskan ke pusat pengolahan data untuk diolah sesuai kebutuhan. Berikut merupakan kelompok AWS dikelompokkan berdasarkan penyajian datanya (Angela, 2017):

1. *Real-time* AWS, yaitu sistem yang penyajian datanya dilakukan secara *real-time*. Sistem model ini dilengkapi dengan sistem komunikasi, berupa alarm untuk memberikan peringatan khusus bila terjadi kondisi cuaca yang ekstrim atau berbahaya, seperti badai, hujan lebat, suhu tinggi dan sebagainya.

2. *Off-time* AWS, yaitu sistem yang hanya merekam data serta menyimpannya pada suatu media penyimpanan. Data yang ditampilkan adalah data aktual. Data yang disimpan tersebut dapat diunduh sewaktu-waktu sesuai keperluan.

2.4 *Fire Weather Index* (FWI)

Indeks cuaca kebakaran atau *Fire Weather Index* merupakan peringkat numerik dari intensitas kebakaran. FWI memiliki beberapa kegunaan antara lain untuk menghitung pengaruh cuaca terhadap bahan bakar hutan, serta untuk mengevaluasi bahaya kebakaran sebagai fungsi dari kondisi cuaca sekarang dan yang lalu. Jika ditinjau dari segi cuaca FWI secara umum dapat disebut sebagai indeks bahaya kebakaran. Bahaya kebakaran adalah indikasi umum dari semua faktor yang mempengaruhi kemudahan terbakar, penyebaran api, dampak fisik kebakaran dan tingkat kesulitan pengendalian kebakaran (Suciarti, 2013)

2.5 Drought Code (DC)

Kode Kekeringan atau yang lebih dikenal dengan *Drought Code* (DC) pertama kali dikembangkan oleh Tunner (1966). DC digunakan sebagai indeks kandungan air yang tersimpan dalam tanah yang dimanfaatkan untuk status kelembaban dari bahan bakar hutan yang mengering secara lambat. Skala DC

umumnya berkisar antara 0-800 dengan ambang batas untuk terjadinya penyalaan antara 400-500. Ambang batas pembasahan untuk DC yaitu, sekitar 2,8 mm di mana pada fase ini kelembaban bahan bakar bertambah setelah terjadinya hujan. Tingkat kejenuhan DC atau kondisi di mana bahan bakar mengandung kelembaban maksimum sebesar 400% (setara dengan 100 mm air) (Wardhana, 2003).

Berbagai metode telah dikembangkan untuk menghitung DC salah satu diantaranya adalah yang dikembangkan oleh *Canadian Forest Service* (CFS). Suhu dan curah hujan merupakan komponen cuaca yang mempengaruhi DC (Wardhana, 2003). Tabel skala tingkat bahaya kebakaran berdasarkan nilai DC dapat dilihat pada Tabel 2.2.

Tabel 2.2 Skala Tingkat Bahaya Kebakaran Berdasarkan Nilai DC

Nilai DC	Skala sifat	Keterangan
0 - 200	Rendah	Mengecek dan persiapan peralatan
201 - 400	Sedang	Staf mengamati ke lapangan
401 - 600	Tinggi	Staf siap siaga bila kebakaran menjadi lebih besar dan luas
601 - 800	Sangat Tinggi	

Sumber : (Wardhana, 2003)

Dari perhitungan DC menggambarkan kemungkinan terjadinya kebakaran yang diekspresikan melalui nilai indeks yang berkisar antara 0 – 800. Nilai DC normal masih berkisar antara 0 - 400 sedangkan diatasnya sudah berbahaya. Skala berdasarkan indeks kekeringan dan potensi terjadinya asap dapat dilihat pada Tabel 2.3.

Tabel 2.3 Indeks Kekeringan dan Potensi Asap

Kelas	Nilai DC	Keterangan
Rendah	0 - 200	Kondisi basah, kabut asap tidak akan terjadi
Sedang	201 - 400	Kondisi normal, kegiatan pembakaran harus dipantau
Tinggi	401 - 600	Kondisi normal puncak musim kering, pembakaran di lahan gambut harus dilarang
Ekstrim	601 - 800	Kondisi bahaya kekeringan, dapat mengakibatkan kabut asap, pembakaran sepenuhnya dilarang

Sumber : (LAPAN, 2020)

Tabel 2.3 dijadikan acuan pada penelitian ini yang mana ketika sistem menampilkan kondisi DC, keterangan dari potensi terjadinya kabut asap juga ditampilkan. Ketika kondisi DC tinggi atau ekstrim Staf Manggala Agni harus dalam kondisi siap siaga. Terutama pada kondisi ekstrim pembakaran harus

dilarang, karena api kecil pun dapat membuat kebakaran yang besar dan menjadikan kawasan tersebut berzona merah.

2.6 Jaringan Syaraf Tiruan

Jaringan syaraf tiruan bisa dibayangkan seperti otak buatan di dalam cerita-cerita fiksi ilmiah. Otak buatan ini dapat berpikir seperti manusia, dan juga sepandai manusia dalam menyimpulkan sesuatu dari potongan-potongan informasi yang diterimanya. Komputer diusahakan agar bisa berpikir sama seperti cara berpikir manusia. Caranya adalah dengan melakukan peniruan terhadap aktivitas-aktivitas yang terjadi di dalam sebuah jaringan syaraf biologis. Ketika manusia berpikir, aktivitas-aktivitas yang terjadi adalah aktivitas mengingat, memahami, menyimpan, dan memanggil kembali apa yang pernah dipelajari oleh otak (Jaya, dkk, 2019).

Jaringan Syaraf Tiruan (JST) adalah sistem pemroses informasi yang memiliki karakteristik mirip dengan jaringan syaraf biologi. Beberapa aplikasi Jaringan Syaraf Tiruan sebagai berikut (Siang, 2005):

1. Pengenalan Pola (*Pattern Recognition*)

Jaringan syaraf tiruan dapat dipakai untuk mengenali pola (misal huruf, angka, suara atau tanda tangan) yang sudah sedikit berubah. Hal ini mirip dengan otak manusia yang masih mampu mengenali orang yang sudah beberapa waktu tidak dijumpainya (mungkin wajah/bentuk tubuhnya sudah sedikit berubah).

2. *Signal Processing*

Jaringan syaraf tiruan (model ADALINE) dapat dipakai untuk menekan *noise* dalam saluran telpon.

3. Peramalan

Jaringan syaraf tiruan juga dapat dipakai untuk meramalkan apa yang akan terjadi di masa yang akan datang berdasarkan pola kejadian yang ada di masa lampau. Ini dapat dilakukan mengingat kemampuan jaringan syaraf tiruan untuk mengingat dan membuat generalisasi dari apa yang sudah ada sebelumnya.

2.7 Normalisasi Data

Sebelum melakukan pelatihan dalam sistem, data yang digunakan harus dinormalisasi terlebih dahulu. Normalisasi bertujuan untuk mendapatkan data

dengan ukuran yang lebih kecil yang mewakili data asli tanpa kehilangan karakteristik sebelumnya. Salah satu metode normalisasi adalah *min-max normalization*, rumus *min-max normalization* data dapat dilihat pada persamaan 2.1 (Budianita, 2013).

$$X^* = \frac{X - \min(X)}{\max(X) - \min(X)} \quad (2.1)$$

dimana :

X* = Data setelah dinormalisasi

X = Data sebelum dinormalisasi

Min(X) = Data terbesar dari himpunan data

Max(X) = Data terkecil dari himpunan data

2.8 MSE (*Mean Square Error*)

MSE merupakan *error* rata-rata kuadrat dari selisih antara *output* jaringan dengan *output* target. Tujuannya adalah memperoleh nilai *error* sekecil-kecilnya secara *iterative* dengan mengganti nilai bobot yang yang terhubung pada semua *neuron* dalam jaringan. Rumus perhitungan MSE dapat dilihat pada persamaan 2.2 (Hermawan, 2014):

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (t_k - y_k)^2 \quad (2.2)$$

dimana :

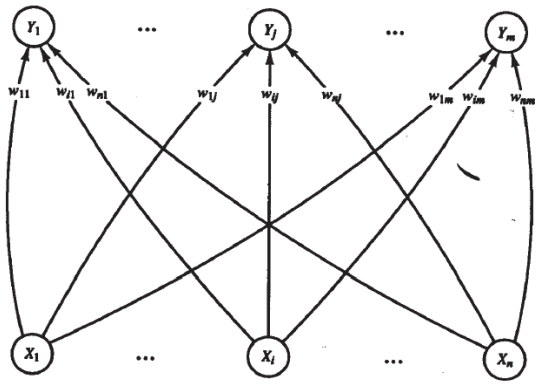
t_k = nilai *output* target

y_k = nilai *output* jaringan

N = jumlah *output* dari target

2.9 LVQ (*Learning Vector Quantization*)

LVQ merupakan suatu metode untuk melakukan pelatihan terhadap lapisan-lapisan kompetitif yang terawasi. Lapisan kompetitif akan belajar secara otomatis untuk melakukan klasifikasi terhadap vektor *input* yang diberikan. Apabila beberapa vektor *input* memiliki jarak yang sangat berdekatan, maka vektor-vektor *input* tersebut akan dikelompokkan dalam kelas yang sama (Kusumadewi, 2004). Contoh arsitektur jaringan LVQ dapat dilihat pada Gambar 2.1.



Gambar 2.1 Contoh Arsitektur Jaringan LVQ

2.10 GLVQ (*Generalized* LVQ)

Metode ini dikembangkan oleh A. Sato dan Yamada pada tahun 1995. Metode ini merupakan pengembangan dari metode *Learning Vector Quantization*, khususnya varian LVQ 2.1, yang sebelumnya telah dikembangkan oleh Kohonen pada tahun 1990. Proses Pelatihan dari metode GLVQ diawali dengan menghitung jarak antara dua vektor menggunakan Euclidean *distance.* Rumus untuk mencari jarak antara data dan bobot dapat dilihat pada persamaan 2.3 (Sato, 1995).

$$d(x_i, w_i) = \sqrt{\sum_{i=1} |x_i - w_i|^2} \quad (2.3)$$

dimana :

$d(x_i, w_i)$ = jarak antara x_i dan w_i

x_i = data ke-i

w_i = bobot ke-i

Selanjutnya didefinisikan perbedaan jarak relatif $\mu(x)$ dengan rumus yang dapat dilihat pada persamaan 2.4 (Sato, 1995) :

$$\mu(x) = \frac{d_j - d_k}{d_j + d_k} \quad (2.4)$$

dimana :

d_j = jarak sesuai kelas sebenarnya atau target

d_k = jarak terkecil, jika jarak terkecil adalah d_j maka diambil jarak terkecil kedua.

$\mu(x)$= perbedaan jarak relatif bernilai -1 sampai 1.

Jika $\mu(x)$ bernilai negatif maka x diklasifikasikan benar, jika $\mu(x)$ bernilai positif maka x diklasifikasikan salah. Dengan demikian ukuran pembelajaran diformulasikan dengan meminimalkan *cost function* (S). Untuk meminimalkan S,

w_1 dan w_2 akan diperbaharui dengan menggunakan persamaan 2.5 dan 2.6 (Sato, 1995).

$$w_{ij} = w_{ij} + \alpha \frac{\partial f}{\partial \mu} \frac{d_k}{(d_j + d_k)^2} (x - w_{ij}) \quad (2.5)$$

$$w_{ik} = w_{ik} - \alpha \frac{\partial f}{\partial \mu} \frac{d_j}{(d_j + d_k)^2} (x - w_{ik}) \quad (2.6)$$

dimana :

α = *learning rate*

w_{ij} = bobot terdekat pada kelas yang sesuai kelas d_j

w_{ik} = bobot terdekat pada kelas yang sesuai kelas d_k

d_j = jarak sesuai kelas sebenarnya atau target

d_k = jarak terkecil, jika jarak terkecil adalah d_j maka diambil jarak terkecil kedua.

$\frac{\partial f}{\partial \mu}$ = turunan dari fungsi sigmoid $f(\mu, t) = \frac{1}{1+e^{-\mu t}}$

2.11 RLVQ (*Relevance* LVQ)

Asumsi data yang diberikan dengan $(\vec{x}^i, y^i) \epsilon \mathbb{R}^n \times \{1, \dots, C\}, i = 1, \dots, p$. Sebuah titik $\vec{x} \in \mathbb{R}^n$ dipetakan ke $y(\vec{w}^i)$ di mana jarak $|\vec{x} - \vec{w}^i|$ adalah minimal. Algoritma pembelajaran mencoba untuk menemukan prototipe seperti yang (hampir) semua titik pelatihan petakan untuk masing-masing label mereka. Inisialisasi prototipe LVQ dengan vektor acak dan beradaptasi secara iteratif dapat dilihat pada persamaan 2.7 (Hammer, 2001).

$$\vec{w}^j \coloneqq \begin{cases} \vec{w}^j + \epsilon \left(\vec{x}^i - \vec{w}^j\right) & jika\ y^i = y(\vec{w}^j) \\ \vec{w}^j - \epsilon \left(\vec{x}^i - \vec{w}^j\right) & lainnya \end{cases} \quad (2.7)$$

Dimana € ∈ (0,1) adalah yang disebut *learning rate*. Subtitusi RLVQ standar euclidian matrik $|\vec{x}^i - \vec{w}^i|$ dalam algoritma di bawah dengan bobot matrik. Untuk mencari jarak antara data dan bobot pada algoritma RLVQ dapat digunakan persamaan 2.8 (Hammer, 2001).

$$|\vec{x}^i - \vec{w}^i|_{\vec{\lambda}} = \left(\sum_k \lambda_k^2 (x_k^i - w_k^j)^2\right)^{\frac{1}{2}} \quad (2.8)$$

Dimana $\lambda_k \geq 0$ merupakan faktor bobot dengan $\sum_k \lambda_k = 1$. Ini diadaptasi dari pembelajaran Hebbian, yaitu pembaruan berulang disertai dengan normalisasi $\vec{\lambda}$, untuk semua k pembaharuan λ_k digunakan persamaan 2.9.

$$\lambda_k = \begin{cases} \lambda_k - \alpha\lambda_k\left(x_k^i - w_k^j\right)^2 & jika\ y^i = y(\vec{w}^j) \\ \lambda_k + \alpha\lambda_k\left(x_k^i - w_k^j\right)^2 & lainnya \end{cases} \quad (2.9)$$

Dimana α > 0 adalah *learning rate* untuk faktor bobot. Ini seperti pembelajaran hebbian karena nilai λ_k terus meningkat yang mana berkontribusi untuk pengelompokkan benar jika normalisasi diperhitungkan (Hammer, 2001).

2.12 GRLVQ (*Generalized Relevance* LVQ)

Hammer dan Villman memperluas GLVQ dengan memasukkan matriks diagonal dengan mengutamakan pentingnya pelatihan bobot pada masing-masing *input* yang disebut relevansi (*relevance*). Istilah GRLVQ diperoleh dari gabungan antara algoritma GLVQ dan algoritma RLVQ. Algoritma GRLVQ memungkinkan suatu klasifikasi berbasis fitur ekstraksi dan mampu menangani dua kendala yaitu masalah bobot *single winning* dan vektor bobot yang *divergence* (Hammer, 2002).

Perumusan GRLVQ dimulai dengan menggabungkan algoritma GLVQ dengan algoritma RLVQ. Persamaan GLVQ yang digunakan adalah persamaan (2.4), (2.5) dan (2.6).

Algoritma RLVQ memperkenalkan vektor relevansi atau bobot *input* $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_n)$, di mana $\lambda_i \geq 0$. Persamaan RLVQ yang digunakan adalah persamaan (2.8). Hammer dan Villman menggunakan ide yang sama seperti algoritma GLVQ dengan menyertakan vektor relevansi. Sehingga algoritma pembaharuan vektor relevansi (bobot input) yang digunakan dalam GRLVQ dapat dilihat pada persamaan (2.10) (Hammer, 2002):

$$\lambda_i = \lambda_i - \alpha\frac{\partial f}{\partial \mu}\left(\frac{d_k}{(d_j+d_k)^2}(x_i - w_{ij})^2 - \frac{d_j}{(d_j+d_k)^2}(x_i - w_{ik})^2\right) \quad (2.10)$$

dimana :

α = *learning rate*

λ_i = bobot *input*

w_{ij} = bobot terdekat pada kelas yang sesuai kelas d_j

w_{ik} = bobot terdekat pada kelas yang sesuai kelas d_k

d_j = jarak sesuai kelas sebenarnya atau target

d_k = jarak terkecil, jika jarak terkecil adalah d_j maka diambil jarak terkecil kedua.

$\frac{\partial f}{\partial \mu}$ = turunan dari fungsi sigmoid $f(\mu) = \frac{1}{1+e^{-\mu}}$

Turunan fungsi sigmoid $f(\mu)$ dapat dilihat pada persamaan 2.11 (Gustiar, 2020):

$$f(\mu)' = \frac{1}{1+e^{-\mu}}\{1 - \frac{1}{1+e^{-\mu}}\} \quad (2.11)$$

Pada algoritma GRLVQ untuk memperbaharui vektor relevansi atau bobot *input* digunakan persamaan 2.10 yang merupakan adaptasi dari algoritma RLVQ. Sedangkan untuk *update* bobot pada algoritma GRLVQ menggunakan persamaan 2.5 dan 2.6 yang diadaptasi dari algoritma GLVQ (Hammer, 2002).

Algoritma GRLVQ :

1. Inisialisasi :
   a. Bobot awal variabel *input* ke-i menuju kelas ke-j : w_{ij}, dengan i = 1,2,...,k; dan j = 1,2,...,m;
   b. Maksimum *epoch*; MaxEpoch.
   c. Vektor relevansi; **λ**.
   d. Parameter *learning rate*; α.
   e. Pengurangan *learning rate;* Decα.
   f. Minimal *learning rate* yang diperbolehkan; Minα.
   g. Minimal *error*; Minerr
2. Selama kondisi berhenti belum terpenuhi, lakukan langkah 3-8.
3. Untuk setiap data *input* x, lakukan langkah 4-7.
4. Untuk setiap x, dihitung nilai d_j yaitu jarak antara x dan w dengan persamaan (2.12).

$$d_j = \sum_{i=1}^{N} \sqrt{\lambda_i (x_i - w_{ij})^2} \quad (2.12)$$

dimana :

d_j = jarak antara x_i dan w_{ij}

λ_i = bobot *input* (vektor relevansi)

x_i = nilai *input*

w_{ij} = bobot

5. Mencari nilai $\mu(x)$ dengan persamaan (2.4)
   Jika $\mu(x)$ bernilai negatif maka x diklasifikasikan benar.

Jika $\mu(x)$ bernilai positif maka x diklasifikasikan salah.

6. Perbaharui bobot (w) dengan ketentuan :
   - Jika x diklasifikasikan benar gunakan persamaan (2.5)
   - Jika x diklasifikasikan salah gunakan persamaan (2.6)
7. Perbaharui vektor relevansi ($\boldsymbol{\lambda}$) dengan persamaan (2.10).
8. Pembaharuan nilai *learning rate* (α) dengan persamaan (2.13) :

$$\alpha_b = \alpha_l - (\alpha_l * Dec\alpha) \quad (2.13)$$

dimana :

α_b = *learning rate* baru

α_l = *learning rate* lama

$Dec\alpha$ = pengurangan *learning rate*

9. Hitung MSE dengan persamaan (2.2)
10. Tes kondisi berhenti : Jika *epoch* ≥ MaxEpoch atau α ≤ minα atau MSE ≤ minerr .

2.13 *Confusion Matrix*

Confusion matrix adalah suatu metode yang biasanya digunakan untuk melakukan perhitungan akurasi pada konsep *data mining*. *Confusion matrix* digambarkan dengan tabel yang menyatakan jumlah data uji yang benar diklasifikasikan dan jumlah data uji yang salah diklasifikasikan (Rahman, 2017). Tabel akurasi dengan *confusion matrix* dapat dilihat pada Tabel 2.4.

Tabel 2.4 Akurasi dengan *Confusion Matrix*

Correct Classification	***Classified as***	
	Predicted "+"	***Predicted "-"***
Actual "+"	*True Positives*	*False Negatives*
Actual "-"	*False Positives*	*True Negatives*

Sumber : (Rahman, 2017)

Berdasarkan tabel *Confusion Matrix* di atas (Rahman, 2017) :

a. *True Positives* (TP) adalah jumlah *record* data positif yang diklasifikasikan sebagai nilai positif
b. *False Positives* (FP) adalah jumlah *record* data negatif yang diklasifikasikan sebagai nilai positif
c. *False Negatives* (FN) adalah jumlah *record* data positif yang diklasifikasikan sebagai nilai positif

d. *True Negatives* (TN) adalah jumlah *record* data negatif yang diklasifikasikan sebagai nilai negatif

Nilai yang dihasilkan melalui metode *Confusion Matrix* adalah berupa evaluasi sebagai berikut (Rahman, 2017) :

a. *Accuracy*, presentase jumlah *record* data yang diklasifikasikan (prediksi) secara benar oleh algoritma. Rumus untuk menghitung akurasi dapat dilihat pada persamaan 2.14.

$$AC = \frac{TP+TN}{Total\ Data} \quad (2.14)$$

b. *Precision***,** rasio dari nilai-nilai positif yang diprediksi dengan benar terhadap total nilai-nilai positif yang diprediksi. Rumus untuk menghitung *precision* dapat dilihat pada persamaan 2.15.

$$P = \frac{TP}{TP+FP} \quad (2.15)$$

c. *Recall*, rasio dari nilai-nilai positif yang diprediksi dengan benar terhadap total nilai-nilai positif yang sebenarnya. Rumus untuk menghitung *recall* dapat dilihat pada persamaan 2.16.

$$R = \frac{TP}{TP+FN} \quad (2.16)$$

d. *Specificity*, persentasi klasifikasi label negatif. Rumus untuk menghitung *specificity* dapat dilihat pada persamaan 2.17.

$$S = \frac{TN}{FP+TN} \quad (2.17)$$

2.14 Laravel

Laravel adalah *framework web* dengan ekspresif *syntax* yang elegan. Selain itu, laravel adalah *framework* yang *clean* dan *classy* untuk pengembangan *web* dengan menggunakan PHP yang dimana juga memiliki *motto* membebaskan *programmer* dari kode yang rumit.

Laravel dirilis dibawah lisensi MIT dengan kode sumber yang sudah disediakan oleh Github, sama seperti *framework-framework* yang lain, Laravel dibangun dengan konsep MVC *(Model-Controller-View)*, kemudian Laravel dilengkapi juga *command line tool* yang bernama “Artisan” yang bisa digunakan untuk *packaging bundle* dan instalasi *bundle* melalui *command prompt* (Aminudin, 2015).

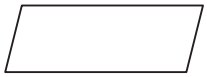
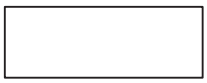
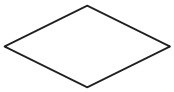
2.15 MySQL

MySQL adalah *database* yang menghubungkan *script* PHP menggunakan perintah *query* dan *escaps character* yang sama dengan PHP. MySQL mempunyai 12 tampilan *client* yang mempermudah dalam mengakses *database* dengan kata sandi untuk mengizinkan proses yang biasa dilakukan. PhpMyAdmin adalah sebuah *software* yang berbentuk seperti halaman situs yang terdapat pada *web server*. Fungsi dari halaman ini adalah sebagai pengendali *database* MySQL sehingga pengguna MySQL tidak perlu repot untuk menggunakan perintah-perintah SQL. Karena dengan adanya halaman ini semua hal tersebut dapat dilakukan hanya dengan meng-klik menu fungsi yang ada pada halaman phpMyAdmin (Saputra, 2013).

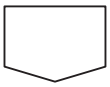
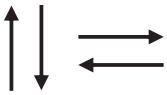
2.16 *Flowchart* (Bagan Alir)

Bagan alir program adalah suatu bagan yang menggambarkan arus logika dari data yang akan diproses dalam suatu program dari awal sampai akhir. Bagan alir program merupakan alat yang berguna bagi programmer untuk mempersiapkan program yang rumit. Bagan alir terdiri dari simbol-simbol yang mewakili fungsi-fungsi langkah program dan garis alir (*flow lines*) menunjukan urutan dari simbol yang akan dikerjakan (Budiman, 2021). Daftar simbol *flowchart* dapat dilihat pada Tabel 2.5.

Tabel 2.5 Daftar simbol *flowchart*

No.	Simbol	Keterangan
1.		Simbol Terminal, simbol yang digunakan untuk menyatakan awal atau akhir suatu program.
2.		Simbol *Input/Output*, simbol yang digunakan untuk menunjukkan operasi masukan atau keluaran
3.		Simbol Proses, simbol yang digunakan untuk menggambarkan proses pengolahan data
4.		Simbol Keputusan, simbol yang digunakan untuk menyatakan suatu pilihan berdasarkan suatu kondisi tertentu

Tabel 2.5 (lanjutan)

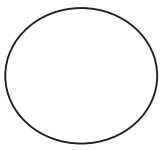
No.	Simbol	Keterangan
5.		Simbol persiapan *(Preparation)*, simbol yang digunakan untuk memberikan nilai awal pada suatu variabel atau pencacah
6.		Simbol proses terdefinisi *(predefined process symbol)*, simbol yang digunakan untuk proses yang detailnya dijelaskan terpisah, misal dalam bentuk *subroutine*
7.		Simbol penghubung ke halaman lain, simbol yang digunakan untuk menghubungkan bagian diagram alir pada halaman yang berbeda
8.		Simbol Penghubung ke halaman yang sama, simbol yang digunakan untuk menghubungkan bagian diagram alir pada halaman yang sama
9.		Simbol Arah aliran, simbol yang digunakan untuk menunjukkan arah aliran proses

Sumber : Budiman, 2021

2.17 *Data Flow Diagram* (DFD)

Data Flow Diagram (DFD) atau dalam bahasa Indonesia menjadi Diagram Alir Data (DAD) adalah representasi grafik yang menggambarkan aliran informasi dan transformasi informasi yang diaplikasikan sebagai data yang mengalir dari masukkan (*input*) dan keluaran (*output*). DFD digunakan untuk merepresentasikan sebuah sistem atau perangkat lunak pada beberapa level abstraksi. DFD dapat dibagi menjadi beberapa level yang lebih detail untuk merepresentasikan aliran informasi atau fungsi yang lebih detail (Rosa, 2018). Notasi-notasi pada DFD (Edward Yourdon dan Tom DeMarco) dapat dilihat pada Tabel 2.6.

Tabel 2.6 Daftar notasi-notasi DFD

No.	Simbol	Keterangan
1.		Proses atau fungsi atau prosedur; pada pemodelan perangkat lunak yang akan diimplementasikan dengan pemrograman terstruktur, maka pemodelan notasi inilah yang harusnya menjadi fungsi atau prosedur di dalam kode program

Sumber : Rosa, 2018

Tabel 2.6 (lanjutan)

No.	Simbol	Keterangan
2.		*File* atau basis data atau penyimpanan *(storage)*; notasi ini digunakan sebagai tabel-tabel basis data yang dibutuhkan, tabel-tabel ini juga harus sesuai dengan perancangan tabel-tabel pada basis data
3.		Entitas luar (*external entity*) atau masukan (*input*) atau keluaran (*output*) atau orang yang memakai / berinteraksi dengan perangkat lunak yang dimodelkan atau sistem lain yang terkait dengan aliran data dari sistem yang dimodelkan
4.		Aliran data; merupakan data yang dikirim antar proses, dari penyimpanan ke proses, atau dari proses ke masukkan (*input*) atau keluaran (*output*)

Sumber : Rosa, 2018