BAB 3
ANALISIS DAN PERANCANGAN

Pada bab ini akan dilakukan analisis dan perancangan terhadap perangkat lunak yang dibangun untuk mengaplikasikan masalah menggunakan metode yang digunakan berdasarkan landasan teori sebelumnya.

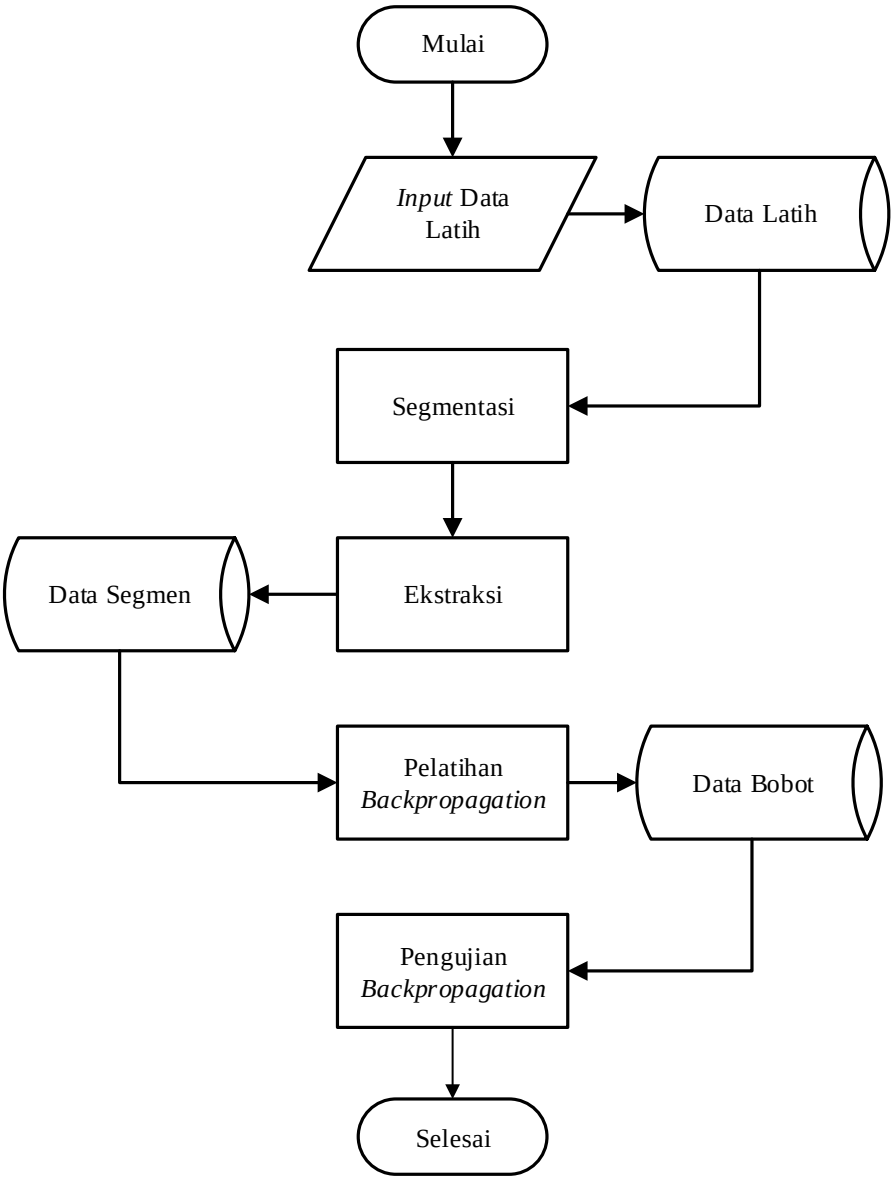
3.1 Analisis Masalah

Gelombang otak menentukan keadaan berpikir serta mental seseorang. Kondisi berpikir seseorang pada dasarnya memiliki dua kondisi berpikir, yaitu rileks dan non-rileks. Rileks merupakan kondisi seseorang yang mampu masuk pada kondisi tenang, kondisi dimana seseorang bisa fokus dan mampu memecahkan permasalahan serta bertindak sebagai gerbang kreativitas seseorang. Selain itu pola gelombang otak manusia yang selalu berubah-ubah menyebabkan kondisi berpikir otak manusia sulit untuk diidentifikasi.

Pembangunan sistem klasifikasi ini diperlukan untuk *monitoring* terhadap kondisi berpikir pada seseorang. Sistem ini dapat mengolah sinyal EEG dengan tahap awal *preprocessing* dan hasil dari proses tersebut akan diklasifikasikan. Keluaran dari sistem ini memberikan informasi berupa kondisi berpikir manusia yaitu rileks dan non-rileks.

3.2 Analisis Sistem

Analisis sistem digunakan untuk mengetahui alur proses dari metode yang digunakan untuk diterapkan pada sistem yang dibangun. Sistem ini memiliki tahap proses dimulai dengan *input* sinyal EEG, *preprocessing* hingga tahap klasifikasi. Alur proses sistem dapat dilihat pada Gambar 3.1.



Gambar 3.1 Alur Proses Sistem

Alur proses sistem pada Gambar 3.1 dimulai dengan memasukkan sinyal EEG yang disimpan dalam bentuk *file* teks (.txt). Perekaman sinyal EEG yang dimasukkan selama 1 menit dengan frekuensi *sampling* 512 Hz. Pada tahap *preprocessing* dibagi menjadi dua tahap yaitu proses segmentasi dan ekstraksi sinyal EEG. Proses segmentasi yaitu dengan membagi sinyal menjadi beberapa segmen selama 2 detik,

sehingga jumlah segmen selama 1 menit menghasilkan 30 segmen. Kemudian tahap kedua pada *preprocessing* yaitu proses ekstraksi sinyal yang sudah di segmentasi terhadap gelombang teta, alfa dan beta. Pada tahap klasifikasi untuk pelatihan dan pengujian menggunakan JST dengan arsitektur *Multi Layer Feedforwards*. *Neuron input* JST berupa sinyal EEG yang sudah diekstraksi terhadap gelombang teta, alfa dan beta. Kemudian proses pelatihan pada JST ini menggunakan algoritma *backpropagation* yang hasil dari pelatihan ini menghasilkan bobot-bobot yang disimpan ke dalam *database* yang digunakan untuk tahapan pada proses pengujian.

3.2.1 Data Masukan

Data masukan untuk sistem klasifikasi kondisi berpikir ini menggunakan data sinyal EEG berbentuk *file* teks (.txt). Perekaman sinyal EEG dilakukan oleh 20 naracoba selama 1 menit terhadap kondisi rileks dan kondisi non-rileks. Penjelasan kondisi naracoba saat perekaman yaitu sebagai berikut [1].

1. Kondisi rileks, naracoba diminta untuk tidak memikirkan apapun, dalam keadaan sadar dengan mata tertutup.
2. Kondisi non-rileks, naracoba diberikan perhitungan matematik sederhana yang dipikirkan atau dikerjakan di dalam hati.

Sinyal EEG yang direkam selama 1 menit dapat dilihat pada Tabel 3.1.

Tabel 3.1 Data Masukan Sinyal EEG Selama 1 Menit

Data ke-i	Waktu (ms)	Data Sinyal (mV)
1	0.00195313	-28
2	0.00390625	-37
3	0.00585938	-38
. . .	. . .	. . .
512	1.00000000	104
. . .	. . .	. . .
30720	60.00000000	57

Sinyal EEG yang dimasukan ke dalam sistem untuk 1 detik menghasilkan data sebanyak 512 data. Sehingga untuk perekaman selama 1 menit menghasilkan data sebanyak 512 data x 60 detik = 30720 data sinyal EEG.

3.2.2 Tahap *Preprocessing*

Tahap *preprocessing* terbagi menjadi dua proses yaitu proses segmentasi dan ekstraksi terhadap gelombang teta, alfa dan beta. Hasil dari proses *preprocessing* akan menghasilkan fitur sinyal EEG yang dibutuhkan untuk data masukan pada tahap klasifikasi.

3.2.2.1 Segmentasi

Segmentasi merupakan proses membagi sinyal menjadi beberapa segmen atau bagian. Sinyal EEG yang dimasukan ke dalam sistem akan dilakukan segmentasi dengan membagi sinyal setiap 2 detik. Segmentasi sinyal dapat dilihat pada Tabel 3.2.

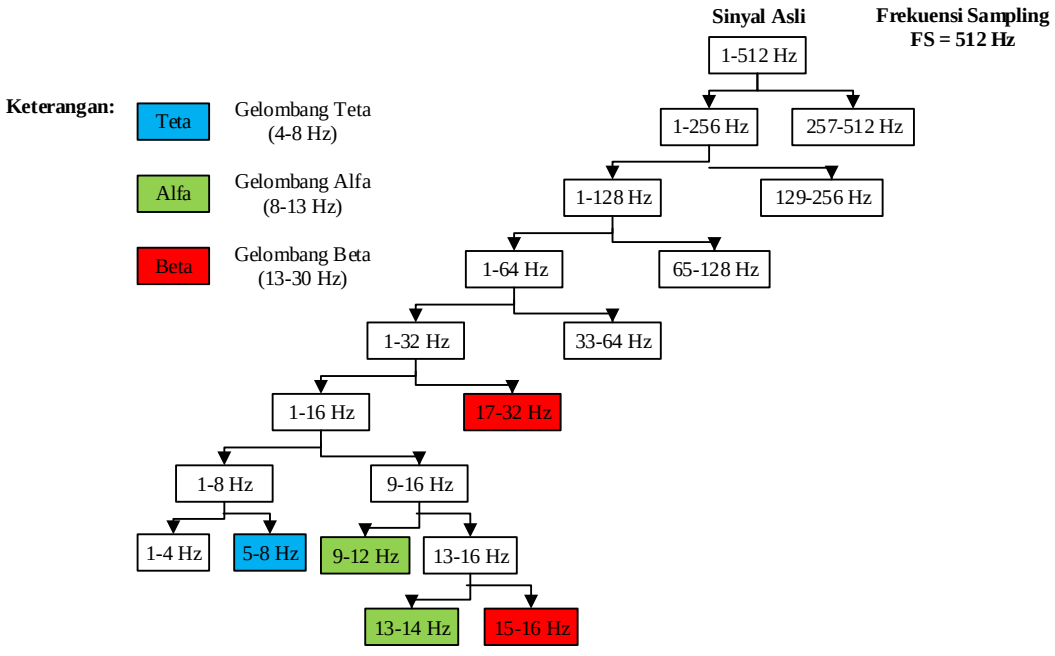
Tabel 3.2 Segmentasi Sinyal EEG Selama 1 Menit

Data ke-i	Waktu (ms)	Data Sinyal (mV)	Segmentasi
1	0.00195313	-28	1
. . .	. . .	. . .	
1024	2.00000000	58	
1025	2.00195313	41	2
. . .	. . .	. . .	
2048	4.00000000	13	
. . .	. . .	. . .	. . .
30720	60.00000000	44	30

Satu detik perekaman menghasilkan 512 data dan dilakukan segmentasi selama 2 detik menghasilkan data sebanyak 1024 data. Sehingga jumlah segmen yang dihasilkan dari 1 menit perekaman akan menghasilkan 30 segmen.

3.2.2.2 Ekstraksi Sinyal

Discrete Wavelet Transform (DWT) digunakan untuk mengekstraksi sinyal EEG untuk mendapatkan fitur ke dalam frekuensi tertentu. Ekstraksi pada DWT menggunakan dekomposisi untuk mereduksi sinyal, proses dekomposisi dibagi menjadi dua proses yaitu konvolusi dan *downsampling*. Seperti pada penelitian sebelumnya, fitur yang digunakan yaitu terhadap gelombang teta, alfa dan beta [1]. Kondisi rileks seseorang didasarkan pada frekuensi rendah, sehingga fitur yang digunakan pada penelitian ini menggunakan gelombang pada daerah teta, alfa dan beta dengan rentang frekuensi 4-30 Hz. Tahapan *dekomposisi* sinyal EEG lebih jelasnya digambarkan dengan pohon sinyal pada Gambar 3.2.



Gambar 3.2 Dekomposisi Sinyal EEG 512 Hz [4]

Berdasarkan Gambar 3.2 arah daun dari pohon sinyal menentukan *filter* yang digunakan. Jika frekuensi berada di cabang kiri maka *filter* yang digunakan adalah *low-pass filter*, sedangkan untuk daun yang berada di cabang kanan menggunakan *high-pass filter*. Gelombang teta dengan frekuensi 4-8 Hz dilakukan dekomposisi sebanyak 7 kali, 6 kali untuk *filter low-pass* dan 1 kali untuk *filter high-pass*. Sehingga sinyal teta yang didapatkan sebanyak 4 data dalam 1 detik dan 8 data dalam 1 segmen. Selanjutnya untuk memperoleh sinyal alfa dalam rentang frekuensi 8-13 Hz dilakukan ekstraksi dalam dua tahap yaitu untuk mendapatkan frekuensi pada rentang 9-12 Hz dan 13-14 Hz yang dilakukan berturut-turut adalah 7 kali dan 8 kali dekomposisi. Ekstraksi yang terakhir adalah untuk mendapatkan gelombang sinyal beta yaitu dengan rentang frekuensi 13-30 Hz dilakukan 2 tahap ekstraksi yaitu terhadap frekuensi 17-32 Hz dan 15-16 Hz berturut-turut sebanyak 4 kali dan 7 kali dekomposisi.

Proses dekomposisi terdapat dua tahap yaitu konvolusi dan *downsampling*. Persamaan (2.3) dan (2.4) masing-masing merupakan konvolusi terhadap *low-pass filter* dan *high-pass filter* yaitu dengan menggunakan koefisien *low pass filter daubechies4 g(n)* (2.1) dan *high-pass filter daubechies4 h(n)* (2.2). Agar lebih mudah dalam proses konvolusi maka panjang koefisien g(n) dan f(n) disamakan dengan panjang data dari sinyal *input*, yaitu dengan menambahkan nilai nol untuk indeks 5 sampai indeks ke-512 sehingga diperoleh:

Low-pass filter:

$$g(n) = [0.48296 \quad 0.83652 \quad 0.27537 \quad -0.07818 \quad 0\ 0\ \dots\ 0] \,, \in \mathrm{R}^{512}$$

High-pass filter:

$$h(n) = [-0.12940 \quad -0.22414 \quad 0.83651 \quad -0.48296 \quad 0\ 0\ \dots\ 0] \,, \in \mathrm{R}^{512}$$

Proses lengkap untuk mendapatkan gelombang sinyal teta, alfa dan beta dari sinyal EEG *input* f(n) diuraikan di bawah ini.

$$f(n) = [-28 \quad -37 \quad -38 \quad \dots \quad 104] \,, \in \mathrm{R}^{512}$$

a. Ekstraksi gelombang teta

Proses ekstraksi terhadap gelombang teta yaitu untuk mendapatkan sinyal frekuensi dengan rentang 4-8 Hz. Untuk mendapatkan frekuensi sinyal pada rentang 4-8Hz, konvolusi 1 sampai dengan 6 menggunakan konvolusi *low-pass filter* dan konvolusi 7 menggunakan *high-pass filter*. Berikut merupakan perhitungan untuk mendapatkan gelombang Teta.

Tabel 3.3 Proses Dekomposisi Gelombang Teta 4-8 Hz

Dekomposisi	Proses
1	Konvolusi *low-pass filter* menggunakan persamaan (2.3): n = 1 $(-28 \ \ x\ 0.48296\ \ ) + (-37 \ \ x\ 0.83652) + \cdots + (104 \ \ x0\ \ ) = -47.68574$ n = 2 $(-37 \ \ x\ 0.48296\ \ ) + (-38 \ \ x\ 0.83652\ \ ) + \cdots + (104 \ \ x0\ \ ) = -53.02371$ . . . n = 512 $(104 \ \ x0.48296\ \ ) = 50.22814$ Hasil Konvolusi: $[-47.68574 \ \ -53.02371 \ \ \ldots \ \ 135.77694 \ \ -50.22814\ ]$ *Hasil Downsampling*: $[-47.68574 \ \ -56.78339 \ \ \ldots \ \ 129.60610 \ \ 135.77694\ ]$
2	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: $[-76.08405 \ \ -85.22973 \ \ \ldots \ \ 174.36177 \ \ 176.17457\ ]$

3	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: $[\ -132.17756 \quad -87.56786 \quad \dots \quad 166.66963 \quad 231.58317\ ]$
4	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: $[\ -134.17107 \quad 22.89371 \quad \dots \quad -81.33100 \quad 274.21835\ ]$
5	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: $[\ -72.90862 \quad -213.50820 \quad \dots \quad 388.62101 \quad 190.10826\ ]$
6	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: $[\ -256.52753 \quad -69.39617 \quad \dots \quad 437.77734 \quad 346.71820\ ]$
7	Konvolusi *high-pass filter* menggunakan persamaan (2.4): n = 1 $(-256.52753 \quad x-0.12940\ ) + (-69.39617 \quad x-0.22414\ ) + \cdots + (437.77734 \quad x0\ ) + (346.71820 \quad x0\ ) = -81.35354$ . . . n = 8 $(346.71820 \quad * -0.12940\ ) = -44.86863$ Hasil Konvolusi: $[-81.35354 \quad 199.74868 \quad \dots \quad -134.36731 \quad -44.86863\ ]$ Hasil *Downsampling*: $[-81.35354 \quad 10.06028 \quad 5.17884 \quad -134.36731\ ]$
Sehingga diperoleh gelombang teta 4-8 Hz: $[-81.35354 \quad 10.06028 \quad 5.17884 \quad -134.36731\ ]$	

Hasil ekstraksi *wavelet* terhadap gelombang teta dari sinyal EEG selama satu detik, mereduksi jumlah data dari 512 sinyal EEG menjadi 4 data gelombang teta.

b. Ekstraksi gelombang alfa

Ekstraksi sinyal EEG terhadap gelombang alfa dengan rentang frekuensi 8-13 Hz yaitu dengan mengambil frekuensi sinyal pada rentang 9-12 Hz dan 13-14 Hz. Berikut merupakan proses untuk mendapatkan frekuensi sinyal pada rentang 9-12 Hz.

Tabel 3.4 Proses Dekomposisi Gelombang Alfa 9-12 Hz

Dekomposisi	Proses
1	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−47.68574 −56.78339 ... 129.60610 135.77694]
2	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−76.08405 −85.22973 ... 174.36177 176.17457]
3	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−132.17756 −87.56786 ... 166.66963 231.58317]
4	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−134.17107 22.89371 ... −81.33100 274.21835]
5	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−72.90862 −213.50820 ... 388.62101 190.10826]
6	Hasil konvolusi *high-pass filter* menggunakan persamaan (2.4) dan *downsampling*: [−102.11273 −260.48913 ... 132.71870 −92.90286]

7	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−187.81356 28.46810 138.07863 −13.61654]
Sehingga diperoleh gelombang alfa 9-12 Hz: [−187.81356 28.46810 138.07863 −13.61654]	

Pada tahap selanjutnya untuk mengambil frekuensi sinyal alfa dengan rentang 13-14 Hz yaitu sebagai berikut.

Tabel 3.5 Proses Dekomposisi Gelombang Alfa 13-14 Hz

Dekomposisi	Proses
1	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−47.68574 − 56.78339 … 129.60610 135.77694]
2	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [− 76.08405 −85.22973 … 174.36177 176.17457
3	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [− 132.17756 −87.56786 … 166.66963 231.58317]
4	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [− 134.17107 22.89371 … −81.33100 274.21835]
5	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [− 72.90862 −213.50820 … 388.62101 190.10826]
6	Hasil konvolusi *high-pass filter* menggunakan persamaan (2.4) dan *downsampling*: [− 102.11273 −260.48913 … 132.71870 − 92.90286]

7	Hasil konvolusi *high-pass filter* menggunakan persamaan (2.4) dan *downsampling*: [367.95054 −419.88316 130.08433 3.64854]
8	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−144.84720 −65.87797]
Sehingga diperoleh gelombang alfa 13-14 Hz: [−144.84720 −65.87797]	

Gelombang alfa merupakan gabungan dari sinyal dengan rentang frekuensi 9-12 Hz sebanyak 4 data dan sinyal dengan rentang frekuensi 13-14 Hz sebanyak 2 data sehingga hasil ekstraksi *wavelet* pada gelombang alfa dari sinyal EEG selama satu detik, mereduksi jumlah data dari 512 menjadi 6 data.

c. Ekstraksi gelombang beta

Ekstraksi sinyal EEG terhadap gelombang beta 13-30 Hz yaitu dengan mengambil sinyal pada frekuensi 17-32 Hz dan 15-16 Hz. Untuk mendapatkan frekuensi sinyal pada rentang 17-32 Hz, dekomposisi 1 sampai dengan 4 prosesnya sama dengan konvolusi pada gelombang teta dan alfa yaitu menggunakan konvolusi *low-pass filter*, kemudian proses konvolusi selanjutnya sebagai berikut.

Tabel 3.6 Proses Dekomposisi Gelombang Beta 17-32 Hz

Dekomposisi	Proses
1	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−47.68574 −56.78339 ... 129.60610 135.77694]
2	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−76.08405 −85.22973 ... 174.36177 176.17457

3	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−132.17756 −87.56786 ... 166.66963 231.58317]
4	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−134.17107 22.89371 ... −81.33100 274.21835]
5	Hasil konvolusi *high-pass filter* menggunakan persamaan (2.4) dan *downsampling*: [−89.50397 171.43499 ... −318.99593 −50.93935]
Sehingga diperoleh gelombang beta 17-32 Hz: [−89.50397 171.43499 ... −318.99593 −50.93935]	

Pada tahap selanjutnya untuk mengambil frekuensi sinyal beta dengan rentang 15-16 Hz yaitu sebagai berikut.

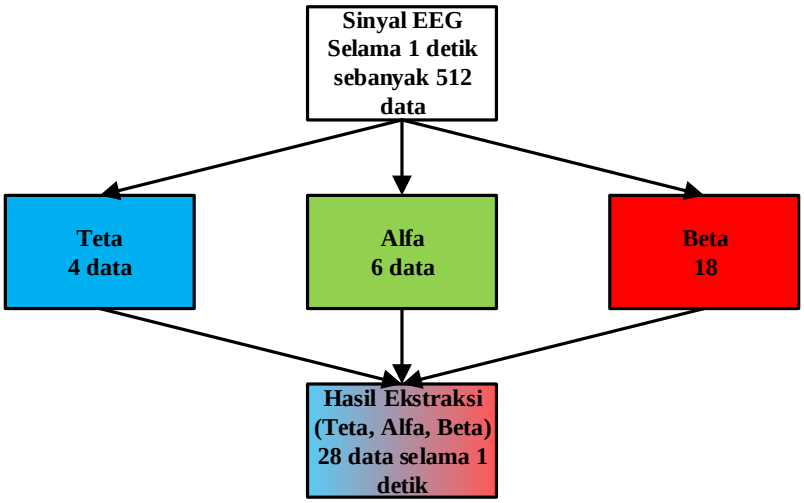
Tabel 3.7 Proses Dekomposisi Gelombang Beta 15-16 Hz

Dekomposisi	Proses
1	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−47.68574 −56.78339 ... 129.60610 135.77694]
2	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−76.08405 −85.22973 ... 174.36177 176.17457
3	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−132.17756 −87.56786 ... 166.66963 231.58317]
4	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: [−134.17107 22.89371 ... −81.33100 274.21835]

5	Hasil konvolusi *low-pass filter* menggunakan persamaan (2.3) dan *downsampling*: $[\ -72.90862 \quad -213.50820 \quad \dots \quad 388.62101 \quad 190.10826\ ]$
6	Hasil konvolusi *high-pass filter* menggunakan persamaan (2.4) dan *downsampling*: $[-102.11273 \quad -260.48913 \quad \dots \quad 132.71870 \quad -92.90286\ ]$
7	Hasil konvolusi *high-pass filter* menggunakan persamaan (2.4) dan *downsampling*: $[\ 367.95054 \quad -419.88316 \quad 130.08433 \quad 3.64854]$
8	Hasil konvolusi *high-pass filter* menggunakan persamaan (2.3) dan *downsampling*: $[\ 153.55348 \quad -17.65195\ ]$
Sehingga diperoleh gelombang beta 15-16 Hz: $[\ 153.55348 \quad -17.65195\ ]$	

Gelombang beta merupakan gabungan dari sinyal dengan rentang frekuensi 17-32 Hz sebanyak 16 data dan sinyal dengan rentang frekuensi 15-16 Hz sebanyak 2 data sehingga hasil ekstraksi *wavelet* pada gelombang beta dari sinyal EEG selama satu detik, mereduksi jumlah data dari 512 menjadi 18 data.

Hasil proses ekstraksi sinyal EEG sebanyak 512 data dalam 1 detik direduksi menjadi 4 data gelombang teta, 6 data gelombang alfa dan 18 data gelombang beta, sehingga hasil ekstraksi digabungkan menjadi 28 data sinyal EEG dapat dilihat pada Gambar 3.3.

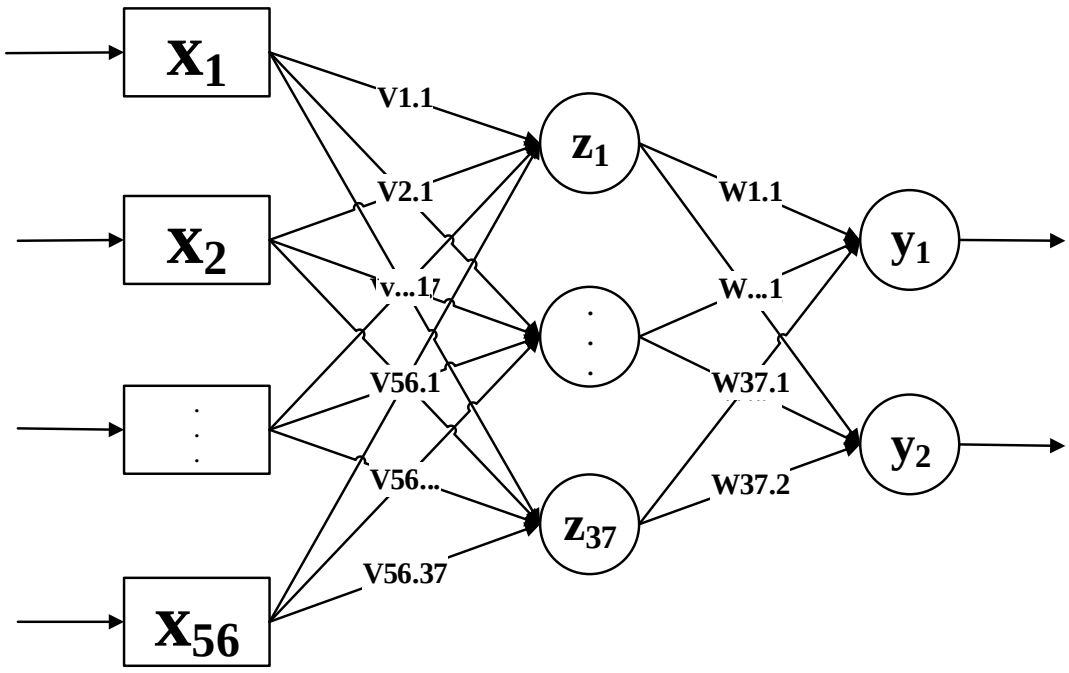


Gambar 3.3 Hasil Ekstraksi Selama 1 Detik

Sinyal EEG yang diperoleh dari hasil ekstraksi tersebut merupakan ekstraksi selama 1 detik, kemudian untuk sinyal EEG yang telah di segmentasi selama 2 detik akan menghasilkan 28 x 2 = 56 data. Maka, data masukan untuk tahap klasifikasi yaitu sebanyak 56 data yang terdiri dari fitur gelombang teta, alfa dan beta.

3.2.3 Pelatihan dan Pengujian *Backpropagation*

Proses yang digunakan pada pelatihan dan pengujian yaitu menggunakan algoritma *backpropagation*. Untuk jumlah *neuron* pada arsitektur *backpropagation* yang digunakan pada penelitian ini dapat dilihat pada Gambar 3.4.



Gambar 3.4 Arsitektur *Backpropagation* yang digunakan

Pada gambar 3.4 terdapat *neuron input layer* (X) dengan jumlah 56 yang merupakan hasil dari proses ekstraksi per segmen, untuk penentuan *neuron* pada *hidden layer* (Z) yaitu 37 *neuron*, ini didasarkan pada penelitian jumlah terhadap *neuron hidden layer* adalah 2/3 dari *neuron input* dan menghasilkan akurasi 70%-90% [16], neuron pada *output layer* (Y) berjumlah 2 *neuron* yang merupakan kelas rileks dan non-rileks. Diantara *neuron input, hidden dan output* terdapat bobot penghubung yaitu bobot antara *input*-hidden layer (V) dan hidden-output (W) serta bobot biasnya (V_0) dan (W_0).

3.2.3.1 Pelatihan *Backpropagation*

Proses yang dilakukan pada tahap pelatihan dengan menggunakan *backpropagation* adalah tahap inisialisasi, proses *feedforward*, *backward* dan *update* bobot.

1. Inisialisasi

Pada penelitian ini jumlah kelas yang digunakan berjumlah dua yaitu terhadap kelas rileks dan non-rileks yang dapat dilihat sebagai matriks pada gambar 3.5. *Parameter* yang digunakan untuk pelatihan yaitu jumlah maksimum *epoch, error minimum* dan laju pembelajaran (*learning rate*). Selanjutnya proses pemberian nilai pada bobot-bobot dengan rentang nilai antara -0.5 sampai 0.5 secara acak yang dapat dilihat sebagai matriks pada Gambar 3.7 dan Gambar 3.8.

Rileks	1	0
Non-Rileks	0	1

Gambar 3.5 Matriks Target Kelas

	Z_1	Z_2	Z_3	…	Z_{37}
X_1	-0.4589	0.3503	0.1253	…	0.4018
X_2	-0.1393	0.1445	-0.0591	…	-0.0524
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
X_{56}	-0.0556	0.1929	0.0911	…	-0.2758
Bias	-0.4118	-0.3554	-0.1310	…	-0.2654

Gambar 3.6 Matriks Bobot V

	Y_1	Y_2
Z_1	-0.2895	-0.0677
Z_2	-0.1059	-0.0946
.	.	.
.	.	.
.	.	.
Z_{37}	-0.4218	-0.4507
Bias	0.4992	0.4600

Gambar 3.7 Matriks Bobot W

2. Proses *feedforward*

Pada tahap *feedforward* merupakan perhitungan nilai pada *hidden* dan *output layer*. Proses lengkap pada tahap *feedforward* akan dijelaskan sebagai berikut.

Tabel 3.8 Proses *Feedforward*

Proses Feedforward
Langkah 1: Setiap *unit input* (X_i, i = 1, 2, 3, …, 56) mengirimkan sinyal *input* yang berada pada *hidden layer*.
Langkah 2: Hitung keluaran di *hidden layer* menggunakan persamaan (2.7). Z_in_1 = -0.4118 + (-4.5174 x -0.4589) + (-20.5236 x -0.1393) + … + (0.86247 x -0.0556) = -0.9161 Z_in_2 = -0.3554 + (-4.5174 x 0.3503) + (-20.5236 x 0.1445) + … + (0.86247 x 0.1929) = 4.1936 . . . Z_in_{37} = -0.2654 + (-4.5174 x 0.4018) + (-20.5236 x -0.0524) + … + (0.86247 x -0.2758) = -6.7523

Langkah 3:

Aktifkan nilai keluaran di *hidden layer* menggunakan persamaan (2.8).

$$Z_1 = \frac{1}{1 + e^{0.9161}} = 0.2857$$

$$Z_2 = \frac{1}{1 + e^{-4.1936}} = 0.9851$$

$$\vdots$$

$$Z_{37} = \frac{1}{1 + e^{6.7523}} = 0.0012$$

Langkah 4:

Hitung keluaran di *output layer* menggunakan persamaan (2.9).

Y_in_1 = 0.4992+ (0.2857 x -0.2895) + (0.9851 x -0.1059) + … + (0.0012 x -0.4218) = 0.5936

Y_in_2 = 0.4600 + (0.2857 x -0.0677) + (0.9851 x -0.0946) + … + (0.0012 x -0.4507) = -0.7659

Langkah 5:

Aktifkan hasil di *output layer* menggunakan Persamaan (2.10).

$$Y_1 = \frac{1}{1 + e^{-0.5936}} = 0.6442$$

$$Y_2 = \frac{1}{1 + e^{0.7659}} = 0.3174$$

3. Proses *Backward*

Pada tahap *backward* merupakan proses menghitung informasi kesalahan dan koreksi bobot yang akan digunakan pada tahap *update* bobot. Proses lengkap *backward* dijelaskan sebagai berikut.

Tabel 3.9 Proses *Backward*

Proses *Backward*
Langkah 1: Menghitung informasi kesalahan di *output layer* menggunakan Persamaan (2.11). δ_1 = (1-0.6442) x (0.6442) x (1-0.6442) = 0.0816 δ_2 = (0-0.3174) x (0.3174) x (1-0.3174) = -0.0688
Langkah 2: Menghitung koreksi bobot dari *hidden layer* ke *output layer* (dengan *learning rate* 0.2) menggunakan Persamaan (2.12). $\Delta W_{1.1}$ = 0.2 x 0.2857 x 0.0816 = 0.0047 $\Delta W_{1.2}$ = 0.2 x 0.9851 x -0.0688 = −0.0136 . . . $\Delta W_{37.2}$ = 0.2 x 0.0012 x -0.0688 = −0.001
Langkah 3: Menghitung koreksi bobot bias dari *hidden layer* ke *output layer* (dengan *learning rate* 0.2) menggunakan Persamaan (2.13). $\Delta W_{0.1}$ = 0.2 x 0.0816 = 0.0163 $\Delta W_{0.2}$ = 0.2 x -0.0688 = −0.0138
Langkah 4: Menghitung nilai delta *input* untuk informasi kesalahan di *hidden layer* menggunakan Persamaan (2.14). $\delta_in_1 = (0.0816 \text{ x } -0.2895) + (-0.0688 \text{ x } -0.0677) = -0.0190$

$\delta_in_2 = (0.0816 \text{ x } -0.1059) + (-0.0688 \text{ x } -0.0946) = -0.0021$

.
.
.

$\delta_in_{37} = (0.0816 \text{ x } -0.4218) + (-0.0688 \text{ x } -0.4507) = -0.0034$

Langkah 5:

Menghitung informasi kesalahan di *hidden layer* menggunakan Persamaan (2.16).

$\delta_1 = -0.0190 \text{ x } -0.9161 \; x \; (1-(-0.9161)) = 0.0333$

$\delta_2 = -0.0021 \text{ x } 4.1936 \; x \; (1-4.1936) = 0.0281$

.
.
.

$\delta_{37} = -0.0034 \text{ x } 6.7523 \; x \; (1-6.7523) = 0.1320$

Langkah 6:

Menghitung koreksi bobot dari *input layer* ke *hidden layer* menggunakan Persamaan (2.17).

$\Delta V_{1.1} = 0.2 \; x \; 0.0333 \text{ x} -4.5174 = -0.0300$

$\Delta V_{1.2} = 0.2 \; x \; 0.0281 \text{ x } -20.5236 = -0.1153$

.
.
.

$\Delta V_{56.37} = 0.2 \text{ x } 0.1320 \text{ x } 0.8254 = 0.0217$

Langkah 7:

Menghitung koreksi bobot bias dari *input layer* ke *hidden layer* menggunakan Persamaan (2.18).

$\Delta V_{0.1} = 0.2 \; x \; 0.0333 = 0.0067$

$\Delta V_{0.2} = 0.2 \; x \; 0.0281 = 0.0056$

. . . $\Delta V_{0.37} = 0.2 \text{ x } 0.1320 = 0.0264$

4. Tahap *Update* Bobot

Pada tahap ini, bobot yang lama yaitu bobot hasil inisialisasi akan digantikan dengan bobot yang baru. Bobot baru merupakan bobot hasil penjumlahan dari bobot lama dengan koreksi bobotnya. Berikut merupakan tahap proses dari *update* bobot. Pada tahap ini, bobot lama beserta biasnya di *update* dengan persamaan.

Tabel 3.10 Proses *Update* Bobot

Proses *Update* Bobot
Langkah 1: Perbaiki bobot W $W_{1.1}(baru) = -0.2895 + 0.0047$ = -0.2848 $W_{1.2}(baru) = -0.0677 - 0.0136$ = -0.0813 . . . $W_{37.2}(baru) = -0.4507 - 0.001$ = -0.4517
Langkah 2: Perbaiki bobot bias W0 $W_{01}(baru) = 0.4992 + 0.0163 = 0.5000$ $W_{02}(baru) = 0.4600 - 0.0138 = 0.4462$
Langkah 3: Perbaiki bobot V $V_{1.1}(baru) = -0.4589 - 0.0300$ = -0.4889 $V_{1.2}(baru) = 0.3503 - 0.1153 = 0.2350$

. . . $V_{56.37}(baru) = -0.2758 + 0.0217 =$ -0.2541
Langkah 4: Perbaiki bobot bias V0 $V_{01}(baru) = -0.4118 + 0.0067 =$ -0.4051 $V_{02}(baru) = -0.3554 + 0.0056 =$ -0.3498 . . . $V_{037}(baru) = -0.2654 - 0.0264 =$ -0.2918

Proses pelatihan akan berhenti bila *Mean Square Error* (MSE) sudah lebih kecil dengan *error* yang sudah ditetapkan. Agar dapat mengendalikan pembelajaran, maka iterasi akan dihentikan jika sudah besar dari *epoch* yang sudah ditetapkan pada tahap inisialisasi. Berikut merupakan bobot dari hasil pelatihan dengan *error 0.001*, *epoch* 1000 dan *learning rate* 0.2.

	Z_1	**Z_2**	**Z_3**	**…**	**Z_{37}**
X_1	-0.4889	0.2350	0.2425	…	-0.3915
X_2	0.2891	0.3812	0.3568	…	-0.0370
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
X_{56}	0.0163	-0.2732	0.1087	…	0.3524
Bias	-0.4051	-0.3498	-0.3723	…	-0.2918

Gambar 3.8 Matriks Bobot V Hasil Pelatihan

	Y_1	Y_2
Z_1	-0.2848	-0.0813
Z_2	0.3669	0.3401
.	.	.
.	.	.
.	.	.
Z_{37}	-0.4770	0.2522
Bias	0.5000	0.4462

Gambar 3.9 Matriks Bobot W Hasil Pelatihan

3.2.3.2 Pengujian *Backpropagation*

Tahap pengujian dengan menggunakan algoritma *backpropagation* hanya menggunakan proses *feedforward*. Bobot yang dihasilkan pada tahap pelatihan akan digunakan untuk proses pengujian. Berikut merupakan proses *feedforward* pada pengujian *backpropagation*.

Tabel 3.11 Proses Pengujian *Feedforward*

Proses *Feedforward*
Langkah 1: Setiap *unit input* (X_i, i = 1, 2, 3, …, 56) mengirimkan sinyal *input* yang berada pada *hidden layer*.
Langkah 2: Hitung keluaran di *hidden layer* menggunakan persamaan (2.7). Z_in_1 = -0.4459 + (-4.5174 x -0.3679) + (-20.5236 x 0.2891) + …+ (0.86247 x -0.3915) = 1.3688 Z_in_2 = -0.4652 + (-4.5174 x -0.1123) + (-20.5236 x 0.3812) + … + (0.86247 x -0.3915) = 0.3521

.
.
.

Z_in_{37} = -0.0716 + (-4.5174 x -0.3915) + (-20.5236 x -0.0370) + … + (0.86247 x 0.3524) = -1.2766

Langkah 3:

Aktifkan nilai keluaran di *hidden layer* menggunakan persamaan (2.8).

$Z_1 = \frac{1}{1+e^{-1.3688}} = 0.7971$

$Z_2 = \frac{1}{1+e^{-0.3521}} = 0.5871$

.
.
.

$Z_{37} = \frac{1}{1+e^{1.2766}} = 0.2181$

Langkah 4:

Hitung keluaran di *output layer* menggunakan persamaan (2.9).

Y_in_1 = 0.1362+ (0.2857 x -0.1308) + (0.9851 x 0.3669) + …+ (0.0012 x -0.4770) = 0.1575

Y_in_2 = -0.0723 + (0.2857 x -0.2477) + (0.9851 x 0.3401) + …+ (0.0012 x 0.2522) = -0.8288

Langkah 5:

Aktifkan hasil di *output layer* menggunakan Persamaan (2.10).

$Y_1 = \frac{1}{1+e^{-0.1575}} = 0.5393$

$Y_2 = \frac{1}{1+e^{-0.8288}} = 0.3038$

Kemudian hasil *output* (Y) akan dibandingkan, jika $Y_0 > Y_1$ maka keluarannya adalah rileks dan juga sebaliknya.

Tabel 3.12 Hasil Pengujian *Backpropagation*

Data/Segmen	S1	S2	S3	…	S30
X_1	-81.3535	-62.7486	303.5434	…	283.0437
X_2	10.0602	-217.6255	-273.9030	…	-427.0816
X_3	5.1788	289.8426	82.6214	…	108.8090
. . .	. . .	. . .	. . .	. . .	. . .
X_{54}	-247.0078	-389.7212	104.8981	…	-71.1654
X_{55}	-105.2351	-167.2512	-11.0013	…	178.6437
X_{56}	212.4528	-284.9532	224.9002	…	-126.9079
Y_0	0.5393	0.8203	0.4783	…	0.5361
Y_1	0.3038	0.2531	0.3467	…	0.4512
Keluaran	rileks	rileks	rileks	…	rileks

3.3 Analisis Kebutuhan Non-Fungsional

Analisis kebutuhan non-fungsional merupakan analisis yang dibutuhkan untuk dapat menentukan spesifikasi dari kebutuhan sistem. Spesifikasi ini meliputi elemen atau perangkat-perangkat yang dibutuhkan untuk sistem yang akan dibangun sampai sistem tersebut dapat diimplementasikan. Analisis kebutuhan ini juga menentukan spesifikasi masukan yang diperlukan sistem, keluaran yang akan dihasilkan sistem dan proses yang dibutuhkan untuk mengolah masukan sehingga dapat menghasilkan suatu keluaran yang diinginkan.

3.3.1 Analisis Perangkat Keras

Perangkat keras komputer atau *hardware* sangat mendukung dalam kinerja sistem, semakin baik spesifikasi perangkat keras yang ada maka kinerja sistem akan semakin baik. Spesifikasi perangkat keras yang dibutuhkan untuk menjalankan sistem klasifikasi ini adalah sebagai berikut.

Tabel 3.13 Spesifikasi Perangkat Keras

No	Komponen	Spesifikasi
1	*Processor*	Intel Pentium 4 1.6 GHz
2	RAM	1 GB
3	VGA	256 MB
4	*Harddisk*	250 GB

3.3.2 Analisis Perangkat Lunak

Perangkat lunak yang digunakan dalam sebuah sistem merupakan perintah-perintah yang diberikan kepada perangkat keras agar saling berinteraksi diantara keduanya. Spesifikasi perangkat lunak yang dibutuhkan untuk menjalankan sistem adalah sebagai berikut.

Tabel 3.14 Spesifikasi Perangkat Lunak

No	Perangkat Lunak	Keterangan
1	Sistem Operasi	Microsoft Windows 7 Ultimate
2	Web Browser	Google Chrome
3	Webserver	XAMPP v.3.2.1

3.3.3 Analisis Pengguna

Adapun kriteria pengguna yang dapat menguji atau menjalankan sistem yang dibangun ini adalah sebagai berikut.

1. Pengguna paham mengoperasikan komputer.
2. Pengguna paham dengan cara kerja sistem klasifikasi.

3.4 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional menggambarkan proses kegiatan yang akan diterapkan dalam sebuah sistem dan menjelaskan kebutuhan yang diperlukan sistem

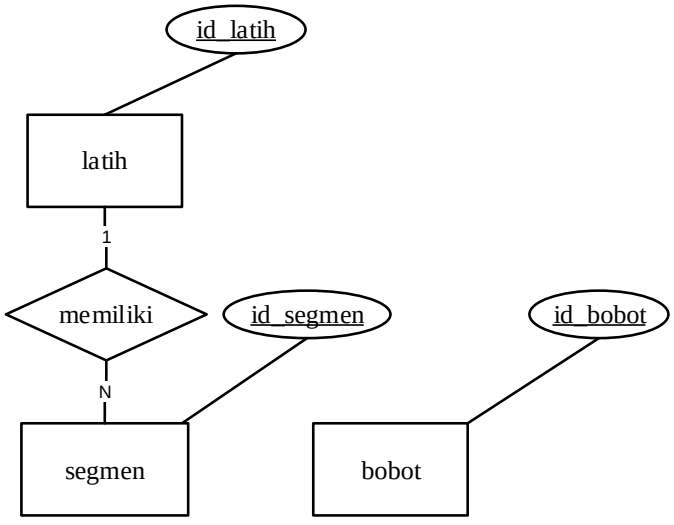
agar sistem dapat berjalan dengan baik sesuai kebutuhan. Analisis kebutuhan fungsional dimodelkan dengan terstruktur.

3.4.1 Analisis Basis Data

Setiap sistem informasi membutuhkan data untuk menghasilkan informasi yang dibutuhkan. Data yang ada saling terhubung atau berelasi antara satu dengan yang lain untuk menyajikan informasi yang benar. Apabila data yang ada tidak berelasi, maka tidak akan terbentuk suatu pengolahan data menjadi informasi untuk memodelkan relasi data dalam model analisis perangkat lunak terstruktur digunakan sebuah alat bantu berupa sebuah diagram yang disebut diagram E-R (*Entity-Relationship*).

3.4.1.1 *Entity Relationship Diagram* (ERD)

ERD adalah model data yang menggunakan beberapa notasi untuk menggambarkan data dalam konteks entitas dan hubungan yang dideskripsikan oleh data tersebut. Diagram hubungan entitas (ERD) menggambarkan sistem yang terdiri dari hubungan antar entitas. Untuk lebih jelasnya ERD yang dibutuhkan untuk sistem yang akan dibangun dapat dilihat pada gambar 3.10.



Gambar 3.10 ERD Sistem Klasifikasi Kondisi Berpikir

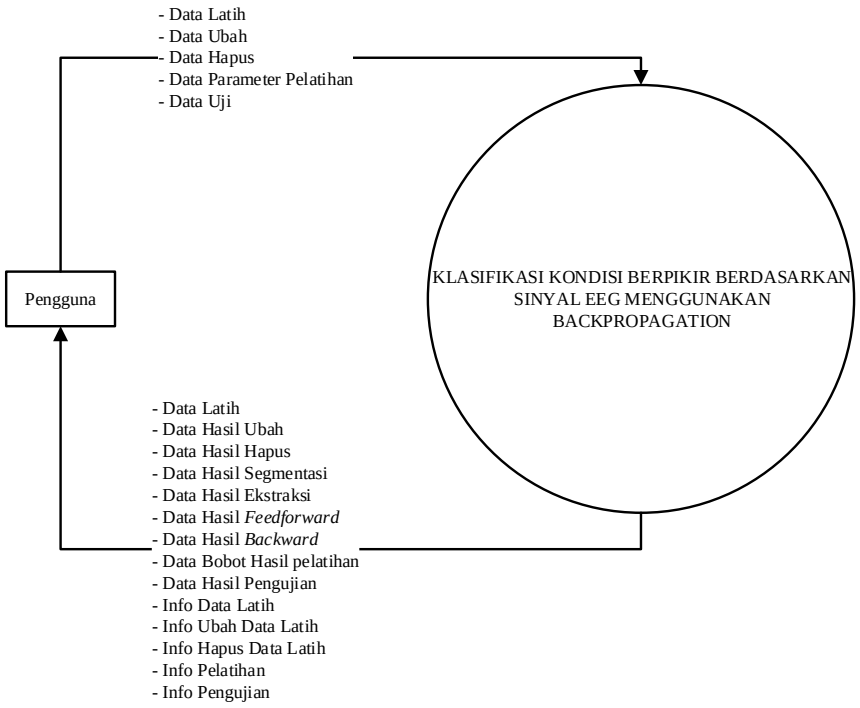
Kamus data dari ERD sistem klasifikasi sinyal EEG ini adalah sebagai berikut.

Tabel 3.15 Kamus Data ERD

Entitas	Atribut
Latih	{id_latih, data_eeg, kelas, naracoba, sampling_rate, kanal}
Segmen	{id_segmen, data_segmen, teta, alfa, beta, id_latih}
Bobot	{id_bobot, nama_bobot, bobot}

3.4.2 Diagram Konteks

Diagram konteks berfungsi untuk menggambarkan hubungan antara entitas luar, masukan dan keluaran sistem, yang direpresentasikan dengan lingkaran tunggal yang mewakili keseluruhan sistem. Diagram Konteks dari sistem klasifikasi ini, dapat di lihat pada gambar 3.11.

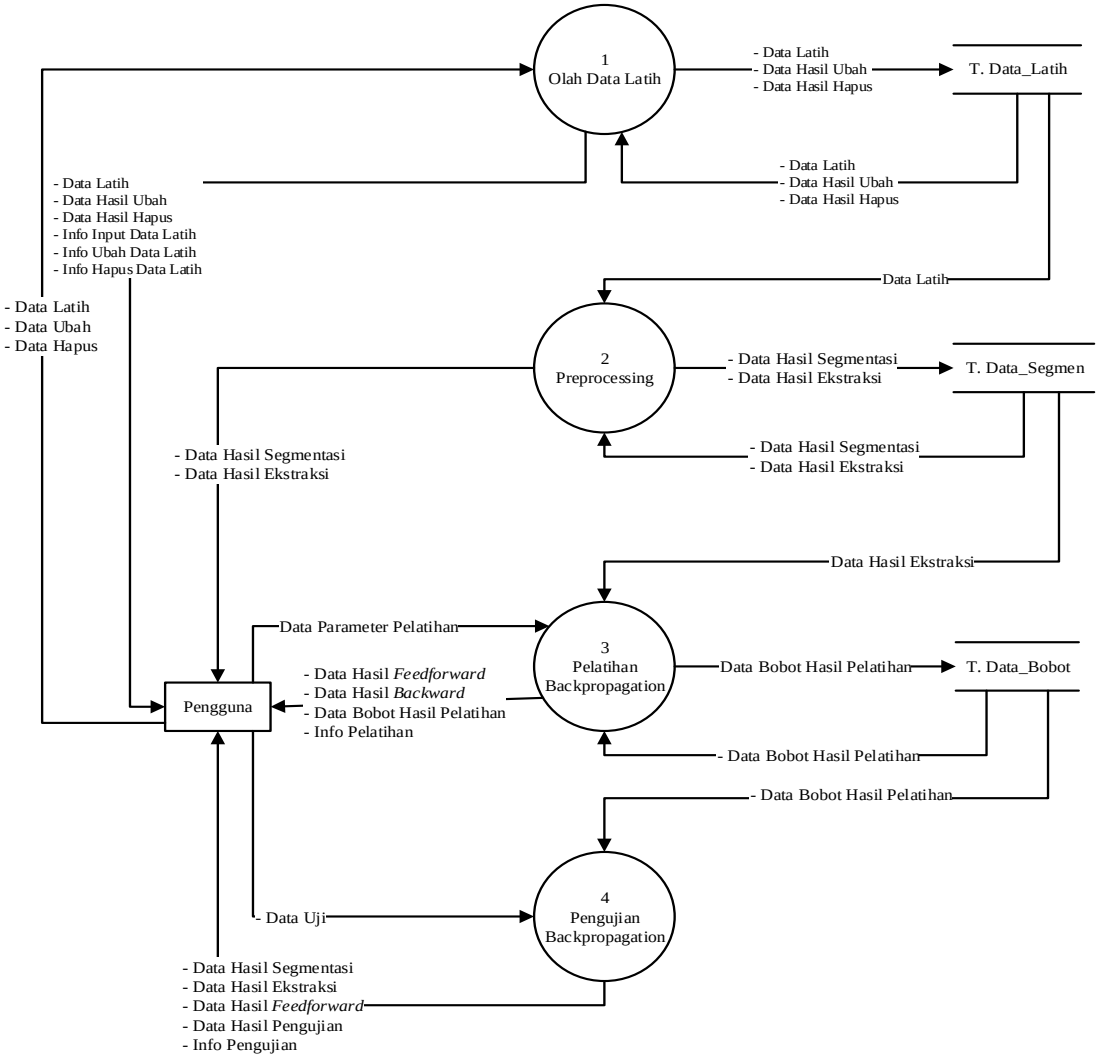


Gambar 3.11 Diagram Konteks Sistem Klasifikasi Kondisi Berpikir

3.4.3 *Data Flow Diagram* (DFD)

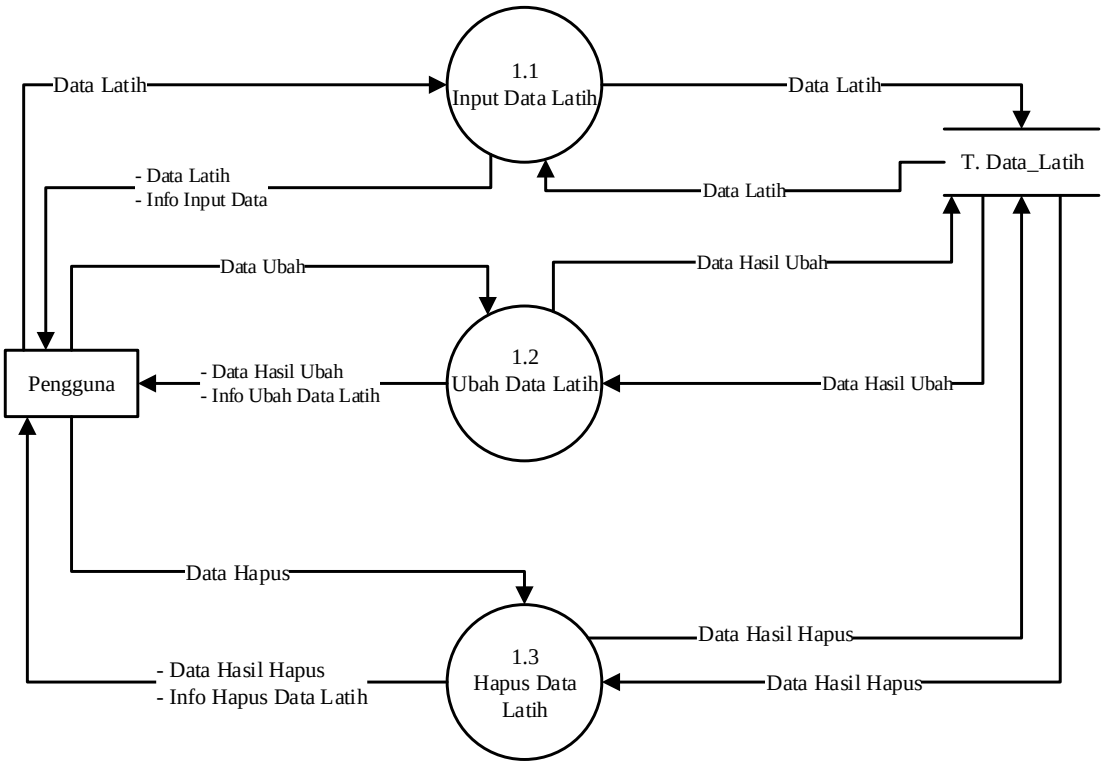
Data flow diagram (DFD) adalah representasi dari suatu sistem yang menggambarkan bagian-bagian dari sistem tersebut beserta seluruh keterlibatan diantara bagian yang ada. Arus data pada (DFD) dapat berupa masukan untuk sistem ataupun keluaran dari sistem, sehingga akan menghasilkan sebuah keluaran yang akan disampaikan kepada pengguna sistem.

3.4.3.1 DFD Level 1 Sistem Klasifikasi Kondisi Berpikir



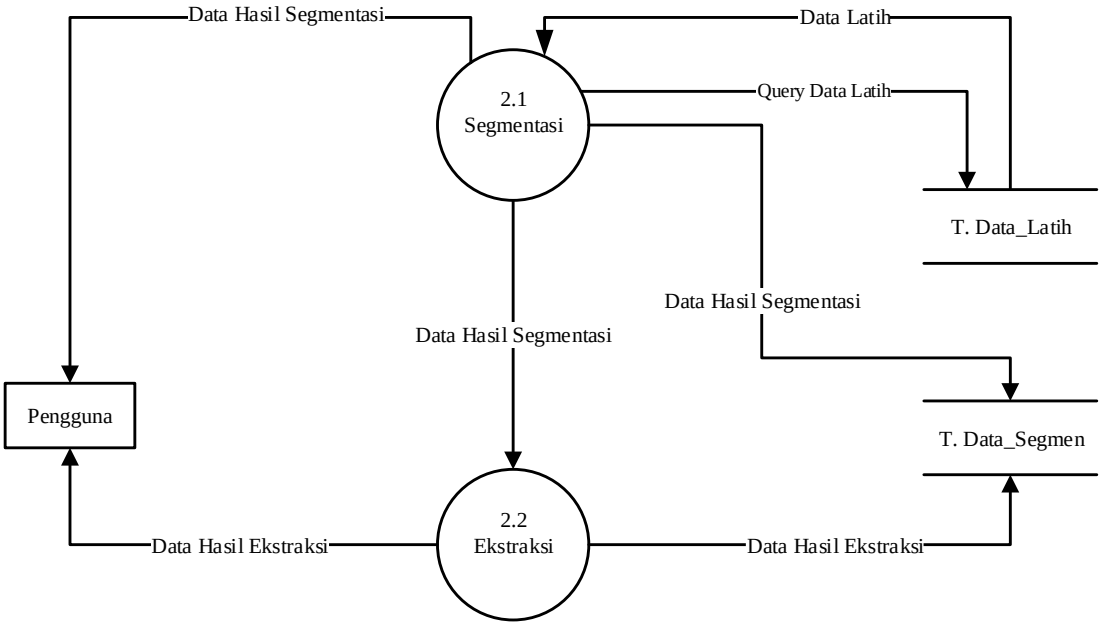
Gambar 3.12 DFD Level 1 Sistem Klasifikasi Kondisi Berpikir

3.4.3.2 DFD Level 2 Proses 1 Olah Data Latih



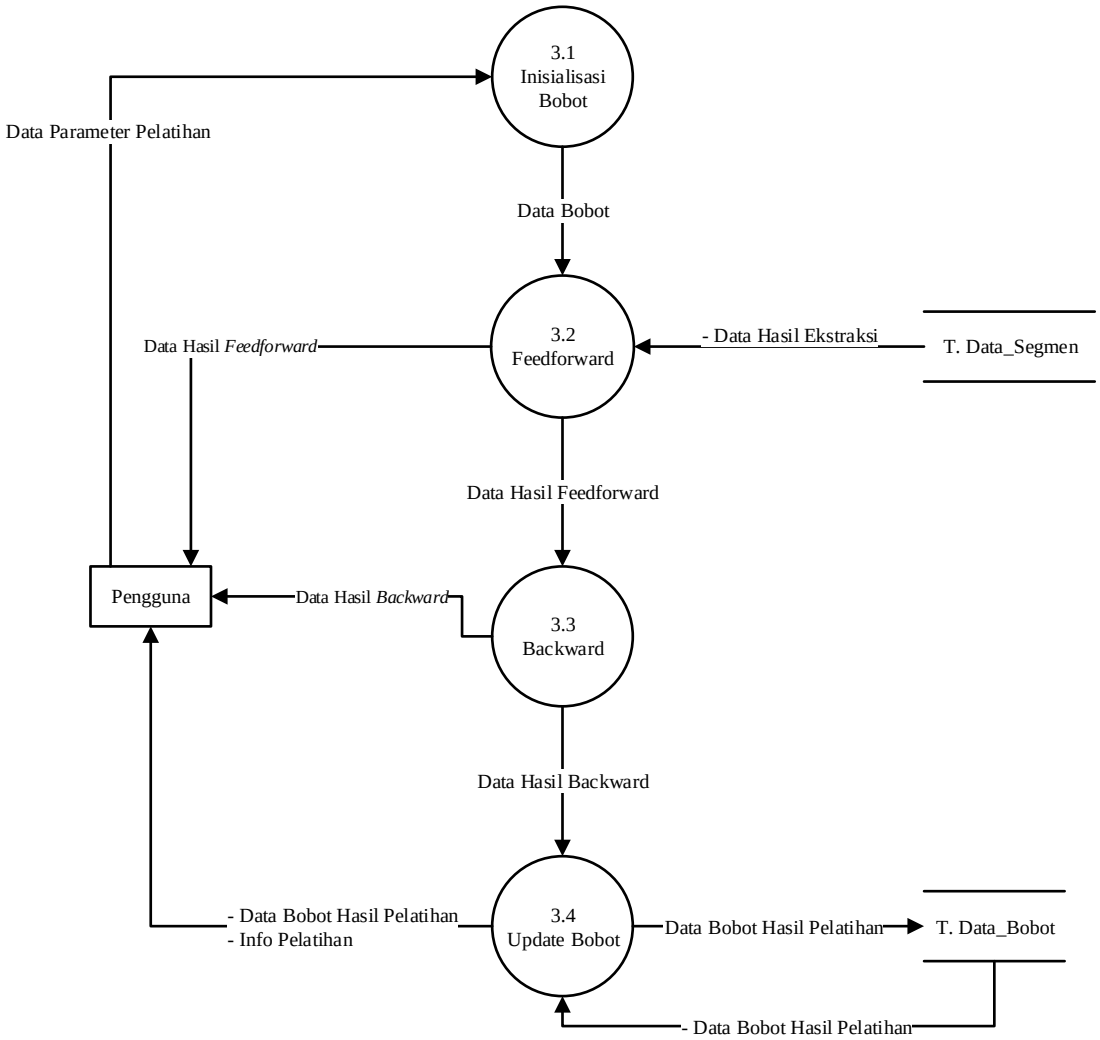
Gambar 3.13 DFD Level 2 Olah Data Latih

3.4.3.3 DFD Level 2 Proses 2 *Preprocessing*



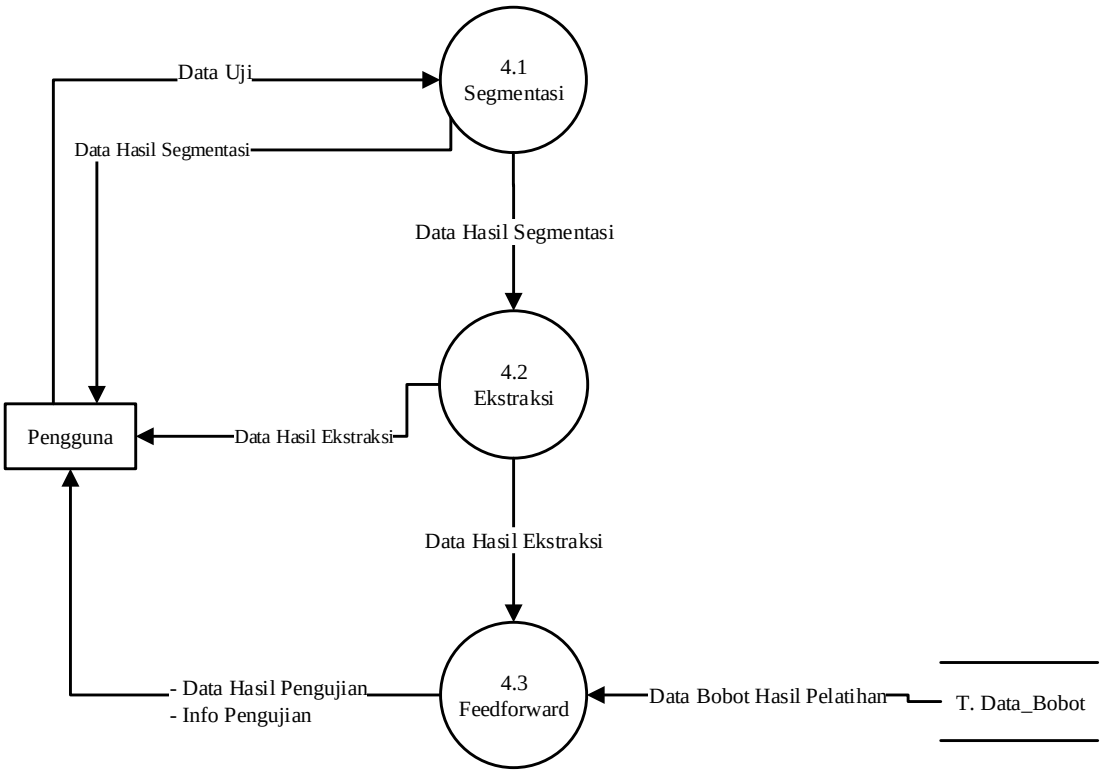
Gambar 3.14 DFD Level 2 *Preprocessing*

3.4.3.4 DFD Level 2 Proses 3 Pelatihan Sistem



Gambar 3.15 DFD Level 2 Pelatihan Sistem

3.4.3.5 DFD Level 2 Proses 4 Pengujian Sistem



Gambar 3.16 DFD Level 2 Pengujian Sistem

3.4.4 Spesifikasi Proses

Spesifikasi proses digunakan untuk menggambarkan semua proses model aliran data yang disajikan pada diagram aliran data. Spesifikasi proses terdapat pada Data Flow Diagram dapat dilihat pada Tabel 3.16.

Tabel 3.16 Spesifikasi Proses

No	Proses	Keterangan
1	No. Proses	1
	Nama Proses	Olah Data Latih
	Sumber	Pengguna
	Masukan	- Data Latih - Data Ubah

		- Data Hapus
	Keluaran	- Data Latih - Data Hasil Ubah - Data Hasil Hapus - Info *Input* Data Latih - Info Ubah Data Latih - Info Hapus Data Latih
	Tujuan	Pengguna
	Logika Proses	1. Pengguna memilih menu olah data latih. 2. Pengguna memilih proses yang terdapat di menu data latih. 3. Sistem melakukan proses sesuai submenu yang dipilih.
No	**Proses**	**Keterangan**
2	No. Proses	1.1
	Nama Proses	*Input* Data Latih
	Sumber	Pengguna
	Masukan	Data Latih
	Keluaran	- Data Latih - Info *Input* Data
	Tujuan	Pengguna
	Logika Proses	1. Pengguna memasukkan data latih. 2. Sistem melakukan proses *input* data latih. 3. Sistem menyimpan data latih ke dalam *data store*. 4. Sistem menampilkan data latih dan info *input* data latih.

No	Proses	Keterangan
3	No. Proses	1.2
	Nama Proses	Ubah Data Latih
	Sumber	Pengguna
	Masukan	Data Ubah
	Keluaran	- Data Hasil Ubah - Info Ubah Data Latih
	Tujuan	Pengguna
	Logika Proses	1. Pengguna memilih data yang akan di ubah. 2. Sistem melakukan proses ubah data latih. 3. Hasil perubahan disimpan ke dalam *data store.* 4. Sistem menampilkan data hasil perubahan dan info perubahan.
No	**Proses**	**Keterangan**
4	No. Proses	1.3
	Nama Proses	Hapus Data Latih
	Sumber	Pengguna
	Masukan	Data Ubah
	Keluaran	- Data Hasil Hapus - Info Hapus Data Latih
	Tujuan	Pengguna
	Logika Proses	1. Pengguna memilih data yang akan di hapus. 2. Sistem melakukan proses hapus data latih. 3. Hasil penghapusan disimpan ke dalam *data store.* 4. Sistem menampilkan data hasil

		penghapusan dan info penghapusan.
No	**Proses**	**Keterangan**
5	No. Proses	2
	Nama Proses	*Preprocessing*
	Sumber	Pengguna
	Masukan	Data Latih
	Keluaran	- Data Hasil Segmentasi - Data Hasil Ekstraksi
	Tujuan	Pengguna
	Logika Proses	1. Pengguna memilih menu *preprocessing*. 2. Pengguna memasukkan data latih. 3. Sistem melakukan proses *preprocessing*. 4. Hasil *preprocessing* disimpan ke dalam data *store.* 5. Sistem menampilkan hasil *preprocessing*.
No	**Proses**	**Keterangan**
6	No. Proses	2.1
	Nama Proses	Segmentasi
	Sumber	Pengguna
	Masukan	Data Latih
	Keluaran	Data Hasil Segmentasi
	Tujuan	- Pengguna - Proses 1.2 Ekstraksi
	Logika Proses	1. Pengguna memasukkan data latih. 2. Sistem melakukan proses segmentasi. 3. Sistem menyimpan hasil segmentasi ke dalam *data store*. 4. Sistem mengirimkan hasil segmentasi ke

		proses 1.2 Ekstraksi dan pengguna.
No	**Proses**	**Keterangan**
7	No. Proses	2.2
	Nama Proses	Ekstraksi
	Sumber	Proses 1.1 Segmentasi
	Masukan	Data Hasil Segmentasi
	Keluaran	Data Hasil Ekstraksi
	Tujuan	Pengguna
	Logika Proses	1. Proses 2.1 Segmentasi mengirimkan hasil segmentasi. 2. Sistem melakukan proses ekstraksi. 3. Sistem menyimpan hasil ekstraksi ke data *store*. 4. Sistem menampilkan hasil ekstraksi.
No	**Proses**	**Keterangan**
8	No. Proses	3
	Nama Proses	Pelatihan *Backpropagation*
	Sumber	Pengguna
	Masukan	Data *Parameter* Pelatihan
	Keluaran	Data Bobot Hasil Pelatihan
	Tujuan	Pengguna
	Logika Proses	1. Pengguna memilih menu pelatihan. 2. Pengguna memasukkan *parameter* pelatihan. 3. Sistem melakukan proses pelatihan. 4. Sistem menyimpan hasil pelatihan ke *data store.* 5. Sistem menampilkan hasil pelatihan ke

		pengguna.
No	**Proses**	**Keterangan**
9	No. Proses	3.1
	Nama Proses	Inisialisasi Bobot
	Sumber	Pengguna
	Masukan	Data Parameter Pelatihan
	Keluaran	Data Bobot
	Tujuan	- Pengguna - Proses 3.2 *Feedforward*
	Logika Proses	1. Pengguna memasukkan *parameter* pelatihan. 2. Sistem melakukan proses inisialisasi bobot. 3. Sistem mengirimkan hasil proses inisialisasi ke proses 3.2 *Feedforward*.
No	**Proses**	**Keterangan**
10	No. Proses	3.2
	Nama Proses	*Feedforward*
	Sumber	Proses 3.1 Inisialisasi
	Masukan	Data Bobot
	Keluaran	Data Hasil *Feedforward*
	Tujuan	- Pengguna - Proses 3.3 *Backward*
	Logika Proses	1. Proses 3.1 Inisialisasi mengirimkan hasil proses inisialisasi berupa data bobot. 2. Sistem mengambil data hasil ekstraksi dari data *store*. 3. Sistem melakukan proses *feedforward*. 4. Sistem mengirimkan hasil *feedforward* ke

		proses 3.3 *Backward*.
No	**Proses**	**Keterangan**
11	No. Proses	3.3
	Nama Proses	*Backward*
	Sumber	Proses 3.2 *Feedforward*
	Masukan	Data Hasil *Feedforward*
	Keluaran	Data Hasil *Backward*
	Tujuan	- Pengguna - Proses 3.4 *Update* Bobot
	Logika Proses	1. Proses 3.2 *Feedforward* mengirimkan data hasil *feedforward.* 2. Sistem melakukan proses *backward*. 3. Sistem mengirimkan hasil *backward* ke proses 3.4 *Update* Bobot.
No	**Proses**	**Keterangan**
12	No. Proses	3.4
	Nama Proses	*Update* Bobot
	Sumber	Proses 3.3 *Backward*
	Masukan	Data Hasil *Backward*
	Keluaran	Data Bobot Hasil Pelatihan
	Tujuan	Pengguna
	Logika Proses	1. Proses 3.3 *Backward* mengirimkan data hasil *backward.* 2. Sistem melakukan proses *update* bobot. 3. Sistem menyimpan hasil pelatihan ke *data store*. 4. Sistem menampilkan data bobot hasil pelatihan ke pengguna.

No	Proses	Keterangan
13	No. Proses	4
	Nama Proses	Pengujian *Backpropagation*
	Sumber	Pengguna
	Masukan	- Data Uji
	Keluaran	- Info Hasil Pengujian
	Tujuan	Pengguna
	Logika Proses	1. Pengguna memilih menu pengujian. 2. Pengguna memasukkan data uji. 3. Sistem melakukan proses pengujian. 4. Sistem menampilkan hasil pengujian ke pengguna.
No	**Proses**	**Keterangan**
14	No. Proses	4.1
	Nama Proses	Segmentasi
	Source (Sumber)	Pengguna
	Input	Data Uji
	Output	Data Hasil Segmentasi
	Destination (Tujuan)	- Pengguna - Proses 4.2 Ekstraksi
	Logika Proses	1. Pengguna memasukkan data uji. 2. Sistem melakukan proses segmentasi. 3. Sistem mengirimkan hasil segmentasi ke proses 4.2 Ekstraksi.
No	**Proses**	**Keterangan**
15	No. Proses	4.2
	Nama Proses	Ekstraksi
	Sumber	Proses 4.1 Segmentasi

	Masukan	Data Hasil Segmentasi
	Keluaran	Data Hasil Ekstraksi
	Tujuan	- Pengguna - Proses 4.3 *Feedforward*
	Logika Proses	1. Proses 4.1 Segmentasi mengirimkan hasil segmentasi. 2. Sistem melakukan proses ekstraksi. 3. Sistem mengirimkan hasil ekstraksi ke proses 4.3 *Feedforward.*
No	**Proses**	**Keterangan**
16	No. Proses	4.3
	Nama Proses	*Feedforward*
	Sumber	Proses 3.2 Ekstraksi
	Masukan	Data Hasil Ekstraksi
	Keluaran	- Info Hasil Pengujian - Data Hasil Pengujian
	Tujuan	Pengguna
	Logika Proses	1. Proses 4.2 Ekstraksi mengirimkan hasil ekstraksi. 2. Sistem mengambil data bobot hasil pelatihan dari *data store*. 3. Sistem melakukan proses *feedforward*. 4. Sistem menampilkan hasil pengujian ke pengguna.

3.4.5 Kamus Data DFD

Kamus data DFD merupakan katalog fakta tentang data dan kebutuhan-kebutuhan informasi dari suatu sistem. Kamus data dibuat berdasarkan arus data yang ada pada diagram aliran data. Dengan menggunakan kamus data, analisis sistem dapat mendefinisikan data yang mengalir pada sistem secara lengkap. Kamus data digunakan untuk merancang *input*, merancang laporan dan *database*. Berdasarkan arus data pada *Data Flow Diagram* yang telah dibuat, maka untuk mengetahui kebutuhan-kebutuhan informasi pada sistem bisa dilihat pada tabel kamus data di bawah ini

1. Data Latih

Tabel 3.17 Kamus Data Latih

Kamus	Keterangan
Nama Aliran	Data Latih
Digunakan pada	- Proses 1 Olah Data Latih - Proses 1.1 Input Data Latih - Proses 2 *Preprocessing* - Proses 2.1 Segmentasi
Deskripsi	Data digunakan pada proses input data latih dan segmentasi.
Struktur Data	sinyal_EEG + naracoba + kelas + kanal + frekuensi_sampling + segmentasi sinyal_EEG = (.txt) naracoba = [‘A-Z’] | [’a-z’] | [0-9] kelas = [‘1 0’|’0 1’] kanal = [‘A-Z’] | [’a-z’] | [0-9] frekuensi_sampling = [0-9] segmentasi = [0-9]

2. Data Ubah

Tabel 3.18 Kamus Data Ubah

Kamus	Keterangan
Nama Aliran	Data Ubah
Digunakan pada	- Proses 1 Olah Data Latih - Proses 1.2 Ubah Data Latih
Deskripsi	Digunakan pada proses ubah data latih.
Struktur Data	id_latih id_latih = [0-9]

3. Data Hapus

Tabel 3.19 Kamus Data Hapus

Kamus	Keterangan
Nama Aliran	Data Hapus
Digunakan pada	- Proses 1 Olah Data Latih - Proses 1.3 Hapus Data Latih
Deskripsi	Digunakan pada proses hapus data latih.
Struktur Data	id_latih id_latih = [0-9]

4. Data Parameter Pelatihan

Tabel 3.20 Kamus Data Parameter Pelatihan

Kamus	Keterangan
Nama Aliran	Data Parameter Pelatihan
Digunakan pada	- Proses 3 Pelatihan *Backpropagation* - Proses 3.1 Inisialisasi Bobot
Deskripsi	Digunakan pada pelatihan *backpropagation* saat proses inisialisasi bobot.

Struktur Data	*epoch* + *error* + learning_rate *epoch* = {0-9} *error* = {0-9} learning_rate = {0-9}

5. Data Uji

Tabel 3.21 Kamus Data Uji

Kamus	Keterangan
Nama Aliran	Data Uji
Digunakan pada	Proses 4 Pengujian *Backpropagation* Proses 4.1 Segmentasi
Deskripsi	Digunakan pada proses pengujian *backpropagation* saat proses segementasi.
Struktur Data	sinyal_EEG = (.txt)

6. Data Hasil Segmentasi

Tabel 3.22 Kamus Data Hasil Segmentasi

Kamus	Keterangan
Nama Aliran	Data Hasil Segmentasi
Digunakan pada	- Proses 2.2 Ekstraksi - Proses 3.1 Ekstraksi
Deskripsi	Data digunakan pada proses ekstraksi.
Struktur Data	data_segmentasi = {0-9}

7. Data Hasil Ekstraksi

Tabel 3.23 Kamus Data Hasil Ekstraksi

Kamus	Keterangan
Nama Aliran	Data Hasil Ekstraksi
Digunakan pada	- Proses 3.2 *Feedforward* - Proses 4.3 *Feedforward*
Deskripsi	Data digunakan pada proses *feedforward.*
Struktur Data	teta + alfa + beta teta = {0-9} alfa = {0-9} beta = {0-9}

8. Data Bobot

Tabel 3.24 Kamus Data Bobot

Kamus	Keterangan
Nama Aliran	Data Bobot
Digunakan pada	- Proses 3.2 *Feedforward*
Deskripsi	Data digunakan pada proses *feedforward.*
Struktur Data	V + V0 + W +W0 V = {0-9} V0 = {0-9} W = {0-9} W0 = {0-9}

9. Data Hasil *Feedforward*

Tabel 3.25 Kamus Data Hasil *Feedforward*

Kamus	Keterangan
Nama Aliran	Data Hasil *Feedforward*
Digunakan pada	Proses 3.3 *Backward*
Deskripsi	Data digunakan pada proses *backward.*
Struktur Data	Y0 + Y1 Y0 = {0-9} Y1 = {0-9}

10. Data Hasil *Backward*

Tabel 3.26 Kamus Data Hasil *Backward*

Kamus	Keterangan
Nama Aliran	Data Hasil *Backward*
Digunakan pada	- Proses 3.4 *Update* Bobot
Deskripsi	Data digunakan pada proses *update* bobot.
Struktur Data	DV + DV0 + DW + DW0 DV = {0-9} DV0 = {0-9} DW = {0-9} DW0 = {0-9}

11. Data Bobot Hasil Pelatihan

Tabel 3.27 Kamus Data Bobot Hasil Pelatihan

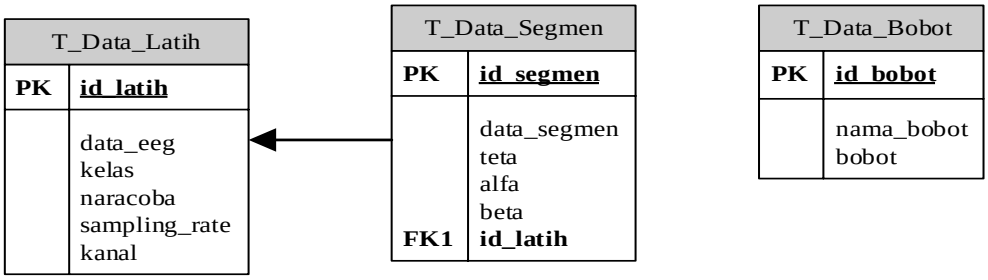
Kamus	Keterangan
Nama Aliran	Data Bobot
Digunakan pada	- Proses 4.3 *Feedforward*
Deskripsi	Data digunakan pada proses *feedforward.*
Struktur Data	V + V0 + W +W0 V = {0-9} V0 = {0-9} W = {0-9} W0 = {0-9}

3.5 Perancangan Basis Data

Perancangan merupakan penggambaran, perencanaan, dan pembuatan sketsa atau pengaturan dari beberapa elemen yang terpisah ke dalam suatu kesatuan yang utuh dan berfungsi. Basis data atau *database* adalah kumpulan *file-file* yang mempunyai kaitan antara satu *file* dengan *file* yang lain.

3.5.1 Skema Relasi

Skema relasi merupakan rangkaian hubungan antara dua tabel atau lebih pada sistem *database*. Pada gambar 3.14 adalah skema relasi pada sistem klasifikasi sinyal EEG.



Gambar 3.17 Skema Relasi Sistem Klasifikasi Sinyal EEG

3.5.2 Struktur Tabel

1. Tabel T_Data_Latih

Tabel 3.28 Struktur Tabel T_Data_Latih

Nama Kolom	Tipe Data	Ukuran	Keterangan
Id_bobot	Int	11	Primary key
Data_eeg	Longtext	-	-
Kelas	Enum	‘1 0’|‘0 1’	-
Naracoba	Int	11	-
Sampling_rate	Int	11	-
Kanal	varchar	3	-

2. Tabel T_Data_Segmen

Tabel 3.29 Struktur Tabel T_Data_Segmen

Nama Kolom	Tipe Data	Ukuran	Keterangan
Id_segmen	Int	11	Primary key
Data_segmen	Longtext	-	-
Teta	Longtext	-	-
Alfa	Longtext	-	-
Beta	Longtext	-	-
Id_latih	Int	11	Foreign key

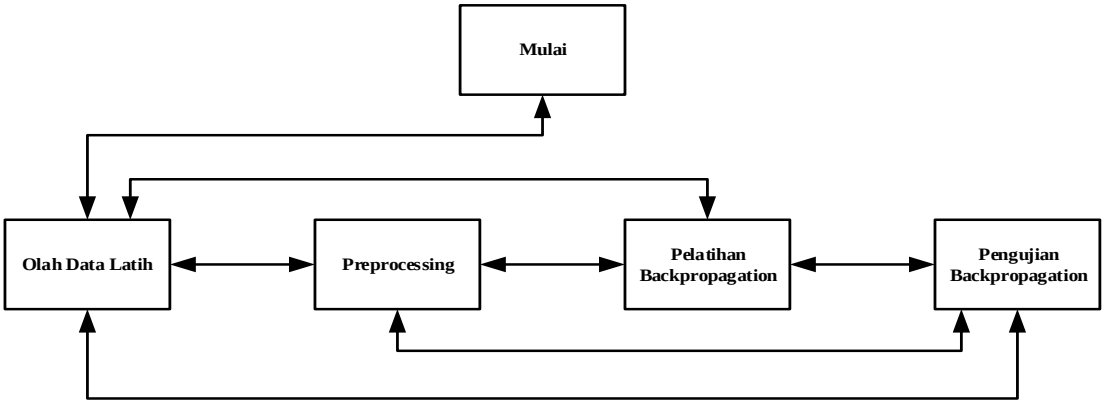
3. Tabel T_Data_Bobot

Tabel 3.30 Struktur Tabel T_Data_Bobot

Nama Kolom	Tipe Data	Ukuran	Keterangan
Id_bobot	int	11	Primary key
Nama_bobot	Char	10	-
Bobot	Int	11	-

3.6 Perancangan Struktur Menu

Struktur menu adalah bentuk umum dari suatu rancangan sistem untuk memudahkan pemakai dalam menjalankan sistem. Sehingga saat menjalankan sistem, pengguna tidak mengalami kesulitan dalam memilih menu yang diinginkan. Pada perancangan perangkat lunak ini dibuat menu seperti dibawah ini.



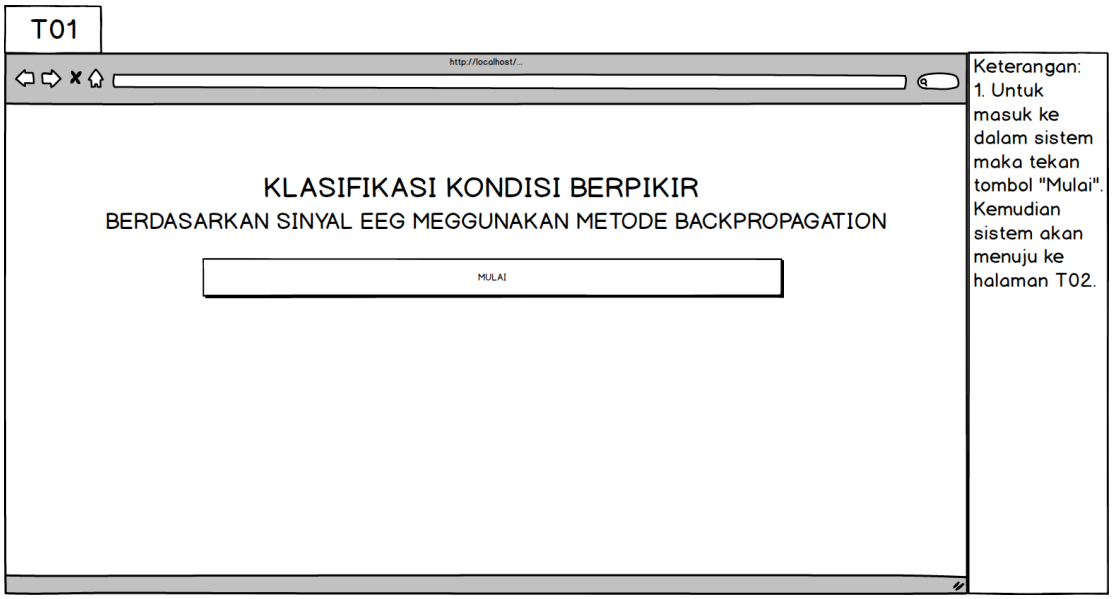
Gambar 3.18 Perancangan Struktur Menu

3.7 Perancangan Antarmuka

Tahap perancangan antarmuka dilakukan untuk merancang antarmuka agar dapat digunakan oleh pengguna. Berikut ini adalah tampilan antarmuka yang dirancang pada sistem klasifikasi.

1. Antarmuka Indeks

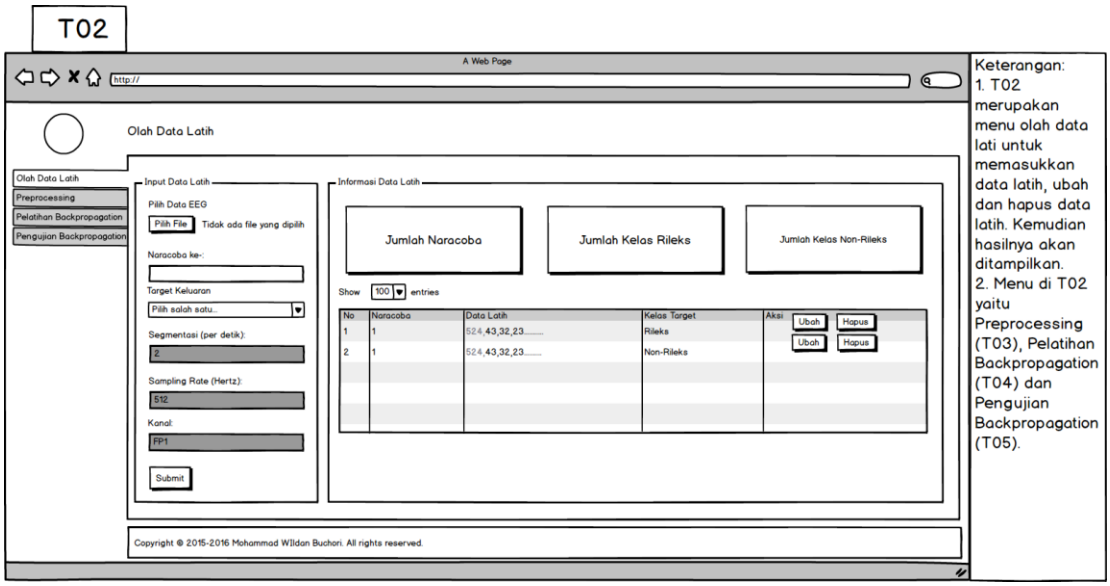
Pada gambar 3.20 merupakan antarmuka halaman indeks, dimana halaman ini hanya menampilkan informasi judul penelitian.



Gambar 3.19 Antarmuka Indeks

2. Antarmuka Olah Data Latih

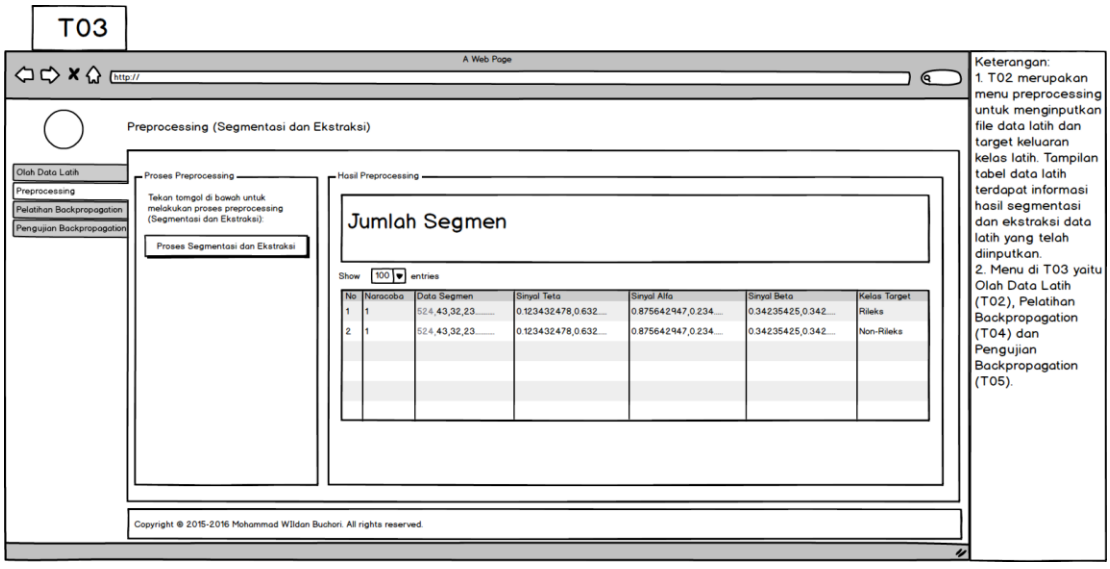
Pada gambar 3.21 merupakan antarmuka halaman olah data latih, dimana halaman ini pengguna memasukkan data latih sinyal EEG dalam *Format* (.txt), target latih kelas dan naracoba. Kemudian, pada halaman ini juga menampilkan informasi sinyal EEG.



Gambar 3.20 Antarmuka Olah Data Latih

3. Antarmuka *Preprocessing*

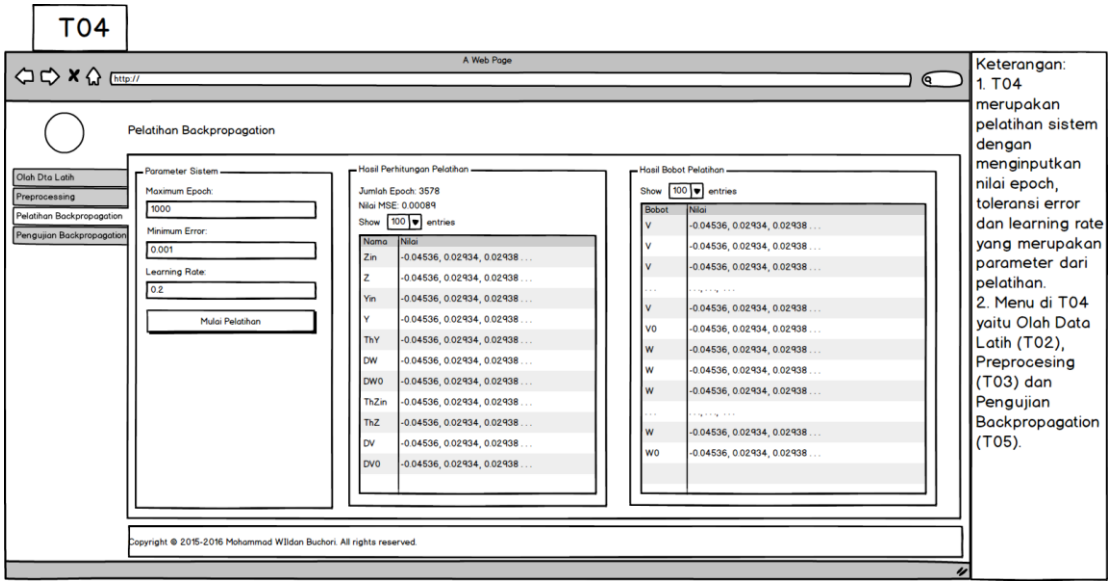
Pada gambar 3.22 merupakan antarmuka halaman *preprocessing* dimana pengguna akan melakukan proses segmentasi dan ekstraksi dengan menekan tombol. Kemudian pada halaman ini juga menampilkan hasil segmentasi dan ekstraksi.



Gambar 3.21 Antarmuka *Preprocessing*

4. Antarmuka Pelatihan *Backpropagation*

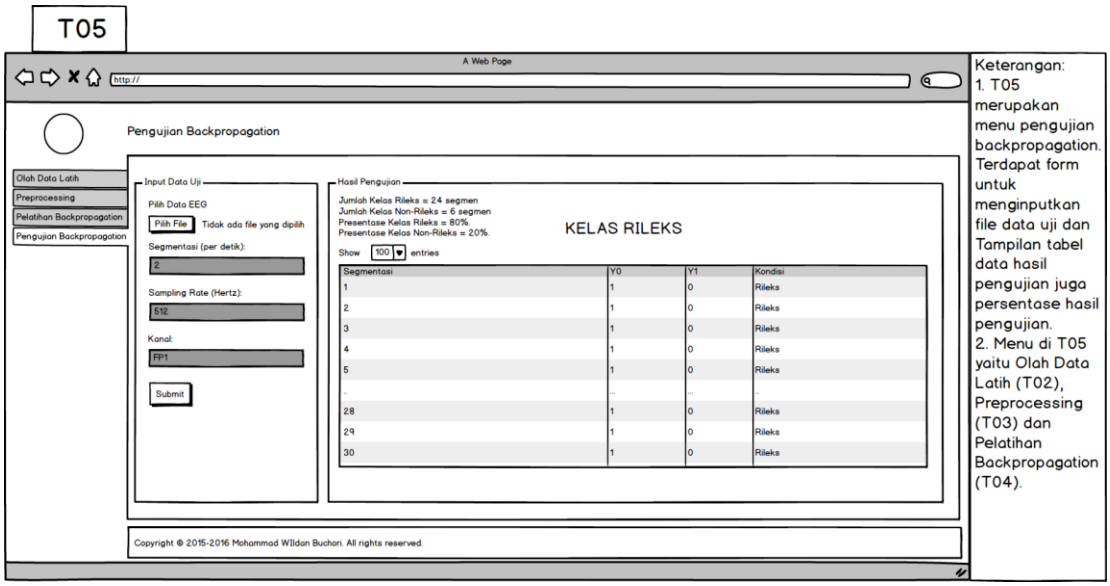
Pada gambar 3.23 merupakan antarmuka halaman pelatihan *backpropagation*, dimana halaman ini pengguna memasukkan *parameter* pelatihan yaitu jumlah *epoch*, *learning rate* dan toleransi *error*. Kemudian, pada halaman ini juga menampilkan informasi bobot hasil pelatihan.



Gambar 3.22 Antarmuka Pelatihan *Backpropagation*

5. Antarmuka Pengujian Backpropagation

Pada gambar 3.24 merupakan antarmuka halaman pengujian *backpropagation*, dimana halaman ini pengguna memasukkan data uji sinyal EEG dalam (.txt), *learning rate* dan toleransi *error*. Kemudian, pada halaman ini juga menampilkan informasi bobot hasil pelatihan.



Gambar 3.23 Antarmuka Pengujian *Backpropagation*

3.8 Perancangan Pesan

Perancangan pesan merupakan informasi kepada pengguna berupa pesan yang ditampilkan kepada pengguna dari aksi yang telah dilakukan. Berikut merupakan perancangan pesan pada sistem ini.

1. Pesan Format Data Latih Tidak Sesuai

Pesan ini akan muncul ketika *format* data latih yang dimasukkan ke dalam sistem bukan dalam bentuk (.txt).



Gambar 3.24 Pesan *Format* Data Latih Tidak Sesuai

2. Pesan *Form* Data Latih Tidak Lengkap

Pesan ini akan muncul jika form yang diisikan oleh pengguna tidak lengkap dan masih ada yang kosong.



Gambar 3.25 Pesan *Form* Data Latih Tidak Lengkap

3. Pesan Sukses Input Data Latih

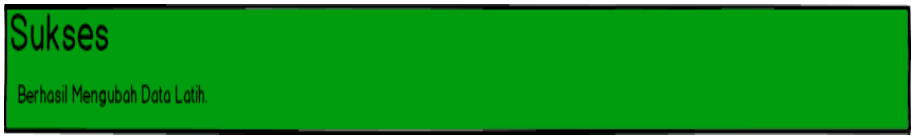
Pesan ini akan muncul ketika proses input data latih sudah selesai dan sukses.



Gambar 3.26 Pesan Sukses Input Data Latih

4. Pesan Sukses Mengubah Data Latih

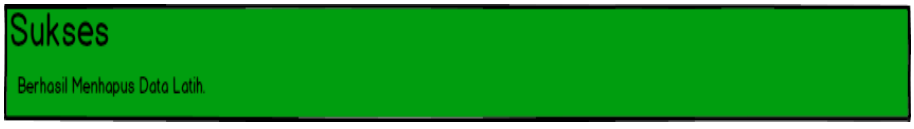
Pesan ini akan muncul ketika mengubah data latih sudah selesai dan sukses.



Gambar 3.27 Pesan Sukses Mengubah Data Latih

5. Pesan Sukses Menghapus Data Latih

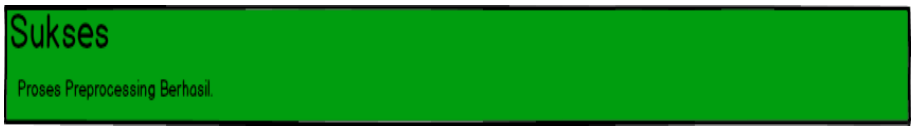
Pesan ini akan muncul ketika menghapus data latih sudah selesai dan sukses.



Gambar 3.28 Pesan Sukses Menghapus Data Latih

6. Pesan Sukses Proses *Preprocessing*

Pesan ini akan muncul ketika proses *preprocessing* sudah selesai dan sukses.



Gambar 3.29 Pesan Sukses Proses *Preprocessing*

7. Pesan Data Latih Masih Kosong

Pesan ini akan muncul ketika data latih masih kosong pada proses *preprocessing.*



Gambar 3.30 Pesan Data Latih Masih Kosong

8. Pesan Proses Pelatihan Sistem

Pesan ini akan muncul ketika proses pelatihan sistem sedang berlangsung.



Gambar 3.31 Pesan Proses Pelatihan Sistem

9. Pesan Data Ekstraksi Masih Kosong

Pesan ini akan muncul ketika data ekstraksi masih kosong pada proses pelatihan.



Gambar 3.32 Pesan Data Ekstraksi Masih Kosong

10. Pesan Format Data Uji Tidak Sesuai

Pesan ini akan muncul ketika *format* data uji yang dimasukkan ke dalam sistem bukan dalam bentuk (.txt).



Gambar 3.33 Pesan *Format* Data Uji Tidak Sesuai

11. Pesan Data Bobot Masih Kosong

Pesan ini akan muncul ketika data bobot masih kosong pada proses pengujian.



Gambar 3.34 Pesan Bobot Masih Kosong

12. Pesan Proses Pengujian Sistem

Pesan ini akan muncul ketika proses pengujian sedang berlangsung.

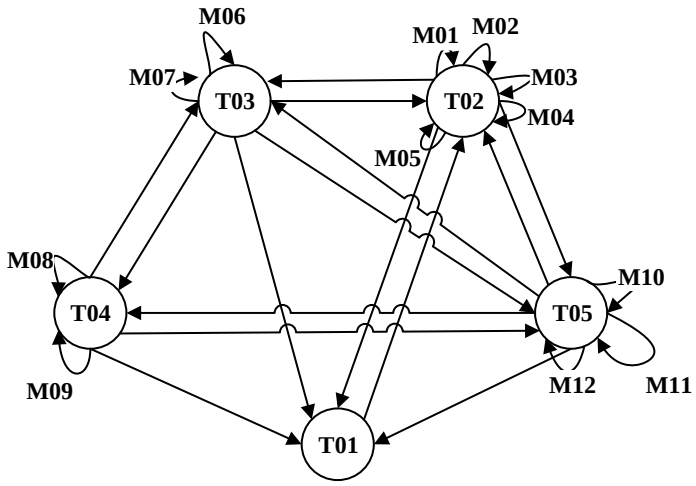


Gambar 3.35 Pesan Proses Pengujian Sistem

3.9 Jaringan Semantik

Jaringan semantik yang terdapat di dalam sistem ini memiliki objek yaitu T01 sampai T05 yang merupakan tampilan dari sistem, kemudian terdapat pesan dari setiap objek tampilan yaitu M01 sampai M05 yang merupakan pesan dari T02 dan juga M06 sampai M07 yang merupakan pesan dari T03, kemudian terdapat pesan dari setiap objek tampilan yaitu M08 sampai M09 yang merupakan pesan dari T04 dan

juga M10 sampai M12 yang merupakan pesan dari T05. Gambar 3.37 merupakan gambaran dari jaringan semantik pada sistem.



Gambar 3.36 Jaringan Semantik

Keterangan:

T01: Antarmuka Indeks

T02: Antarmuka Olah Data Latih

T03: Antarmuka *Preprocessing*

T04: Antarmuka Pelatihan *Backpropagation*

T05: Antarmuka Pengujian *Backpropagation*

M01: Pesan *Format* Data Latih Tidak Sesuai

M02: Pesan *Form* Data Latih Tidak Lengkap

M03: Pesan Sukses *Input* Data Latih

M04: Pesan Sukses Mengubah Data Latih

M05: Pesan Sukses Menghapus Data Latih

M06: Pesan Sukses Proses *Preprocessing*

M07: Pesan Data Latih Masih Kosong

M08: Pesan Proses Pelatihan Sistem

M09: Pesan Data Ekstraksi Masih Kosong

M10: Pesan Format Data Uji Tidak Sesuai

M11: Pesan Data Bobot Masih Kosong

M12: Pesan Proses Pengujian Sistem

3.10 Perancangan Algoritma

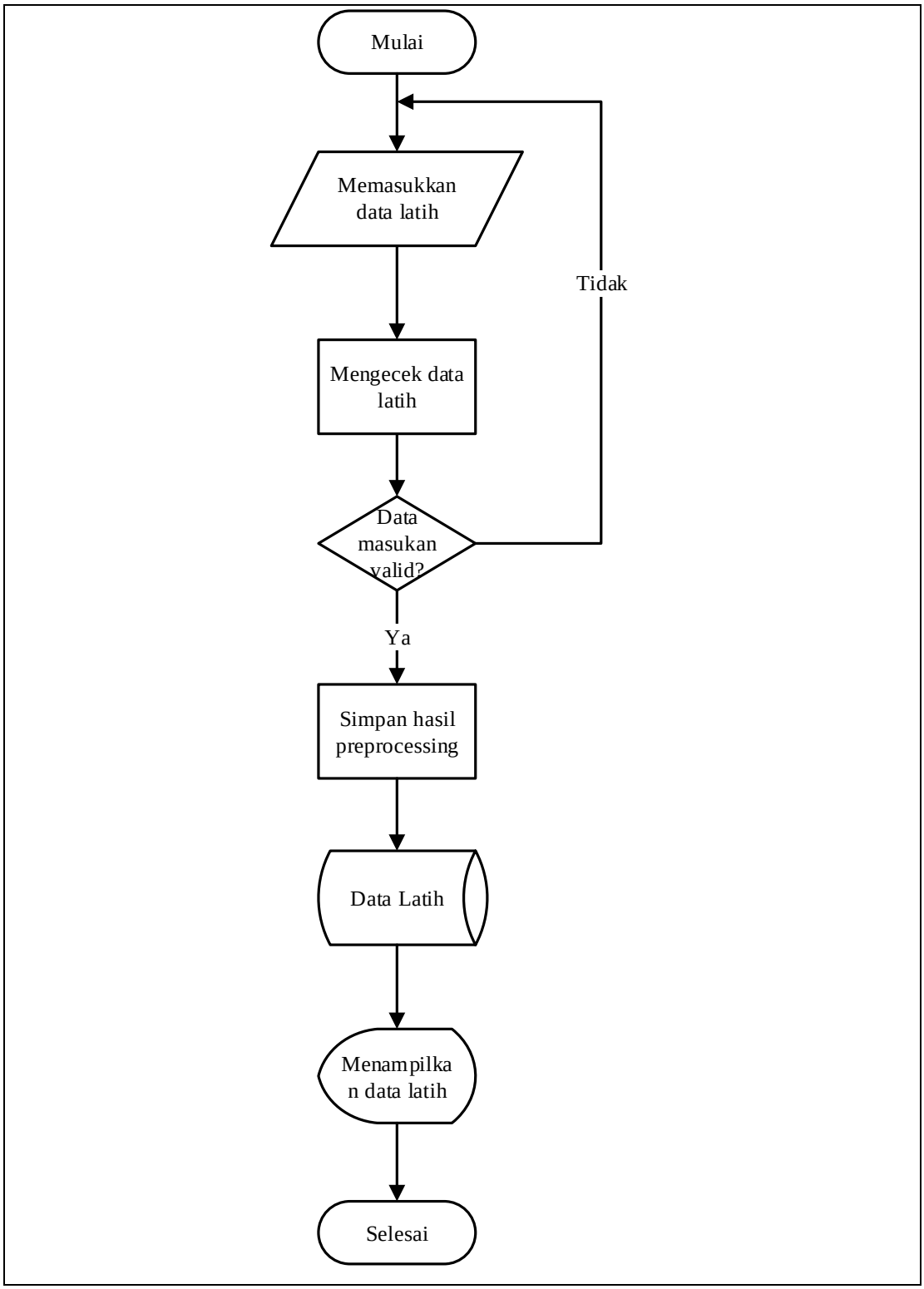
Pada penelitian ini algoritma dijelaskan dalam bentuk *flowchart* dan *pseudocode*. Berikut merupakan penjelasan algoritma yang digunakan pada penelitian ini.

3.10.1 *Flowchart*

Flowchart ini digunakan sebagai algoritma dasar dalam mengkodekan prosedur yang ada. Membuat *flowchart* tujuannya untuk menggambarkan suatu tahapan penyelesaian masalah, secara sederhana, terurai, rapi, dan jelas menggunakan simbol-simbol standar. Adapun perancangan prosedural untuk pembangunan sistem klasifikasi yang akan dibangun adalah sebagai berikut.

1. *Flowchart Input* Data Latih

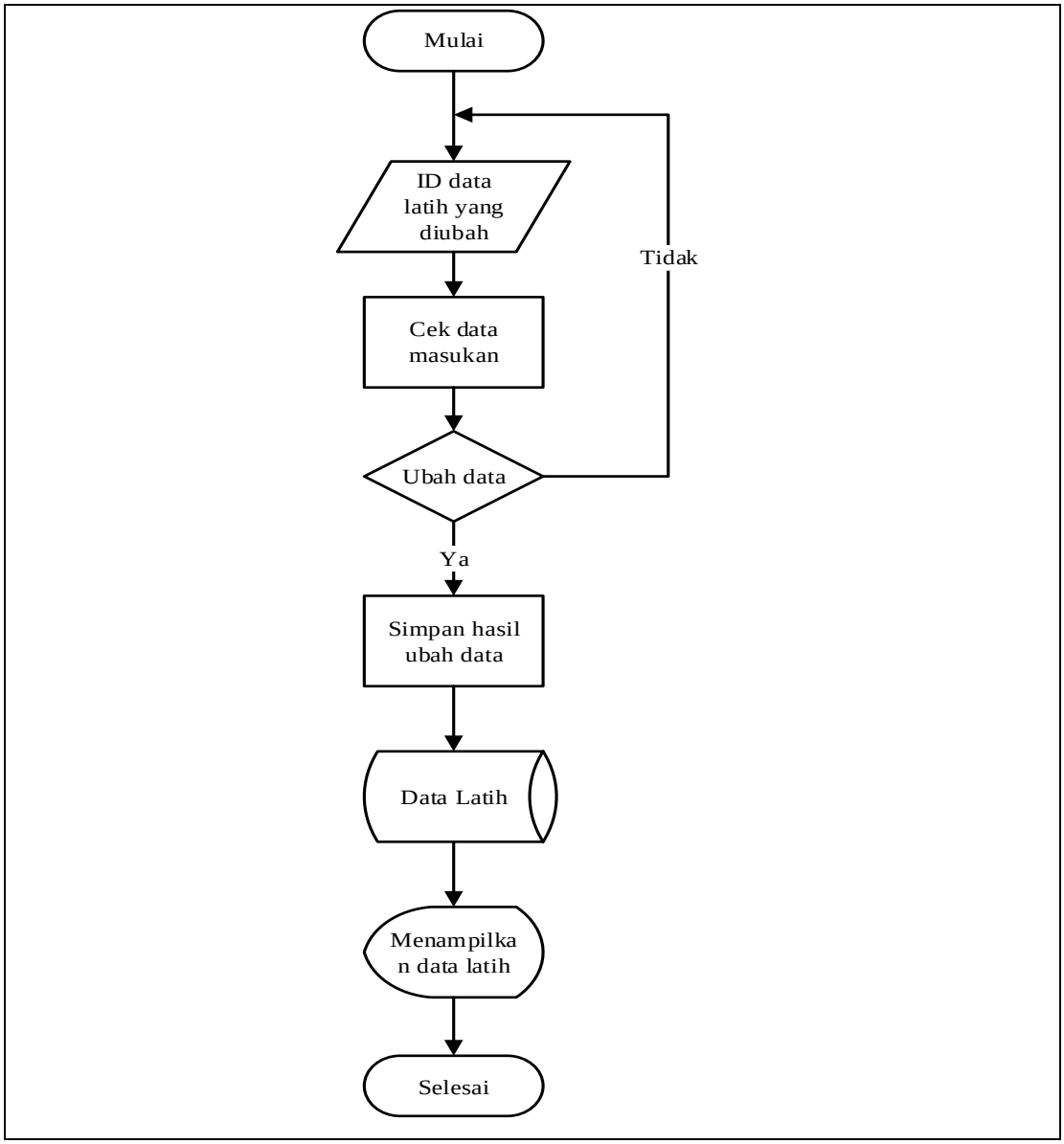
Flowchart ini merupakan prosedur yang terjadi ketika pengguna akan melakukan proses memasukkan data latih. Dimulai dengan memasukkan data latih sinyal EEG, kemudian sistem akan mengecek data yang dimasukkan. Jika data valid maka akan dilakukan proses *input* data latih dan kemudian hasilnya akan disimpan ke dalam *database* serta ditampilkan.



Gambar 3.37 *Flowchart Input* Data Latih

2. *Flowchart* Ubah Data Latih

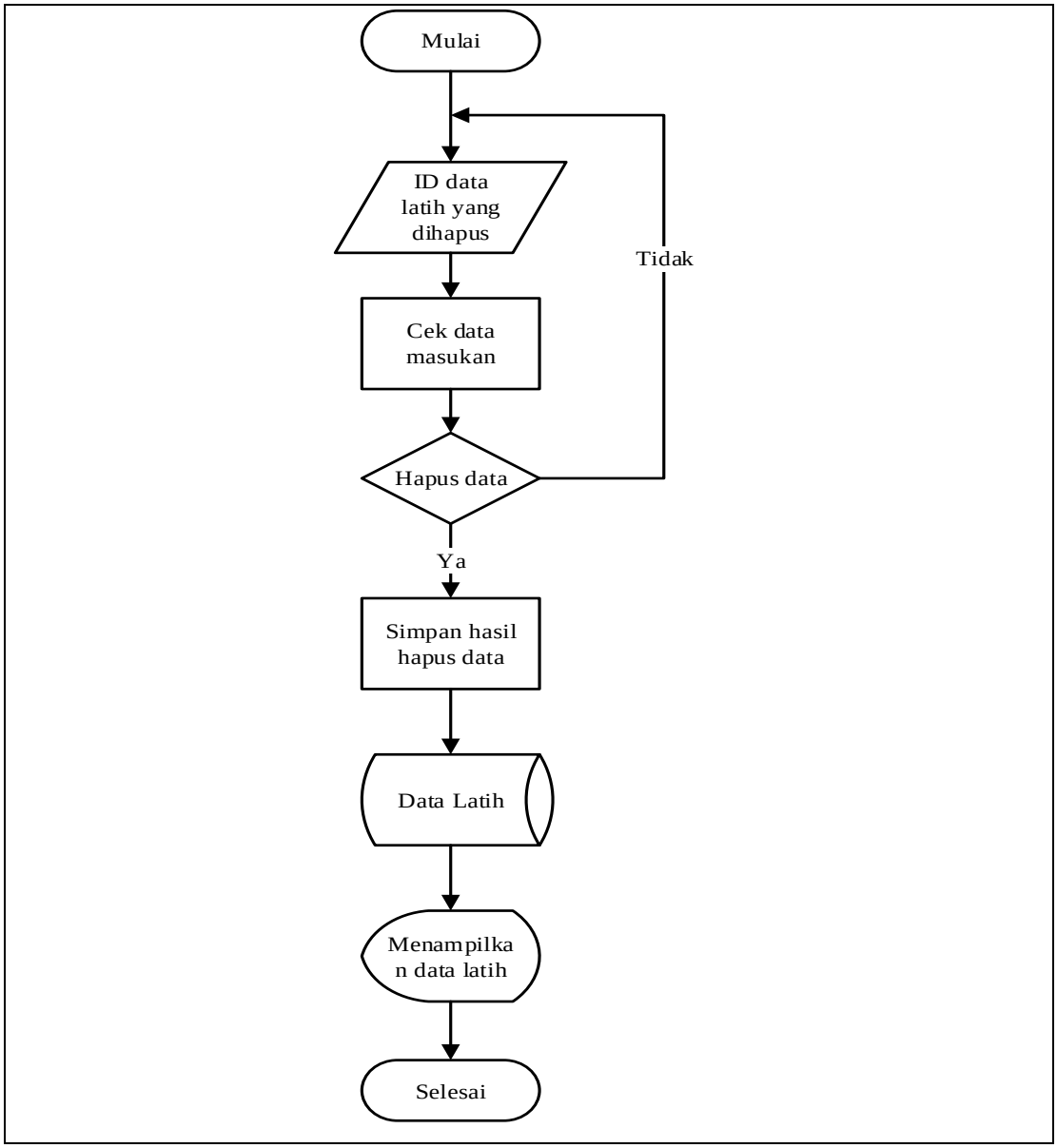
Flowchart ini merupakan prosedur yang terjadi ketika pengguna akan melakukan proses perubahan pada data latih. Dimulai dengan memilih data latih yang akan dirubah. Kemudian sistem akan melakukan perubahan sesuai dengan data yang telah dimasukkan. Jika data akan dirubah maka sistem akan menyimpan perubahan ke dalam *database* dan kemudian perubahan akan ditampilkan.



Gambar 3.38 *Flowchart* Ubah Data Latih

3. *Flowchart* Hapus Data Latih

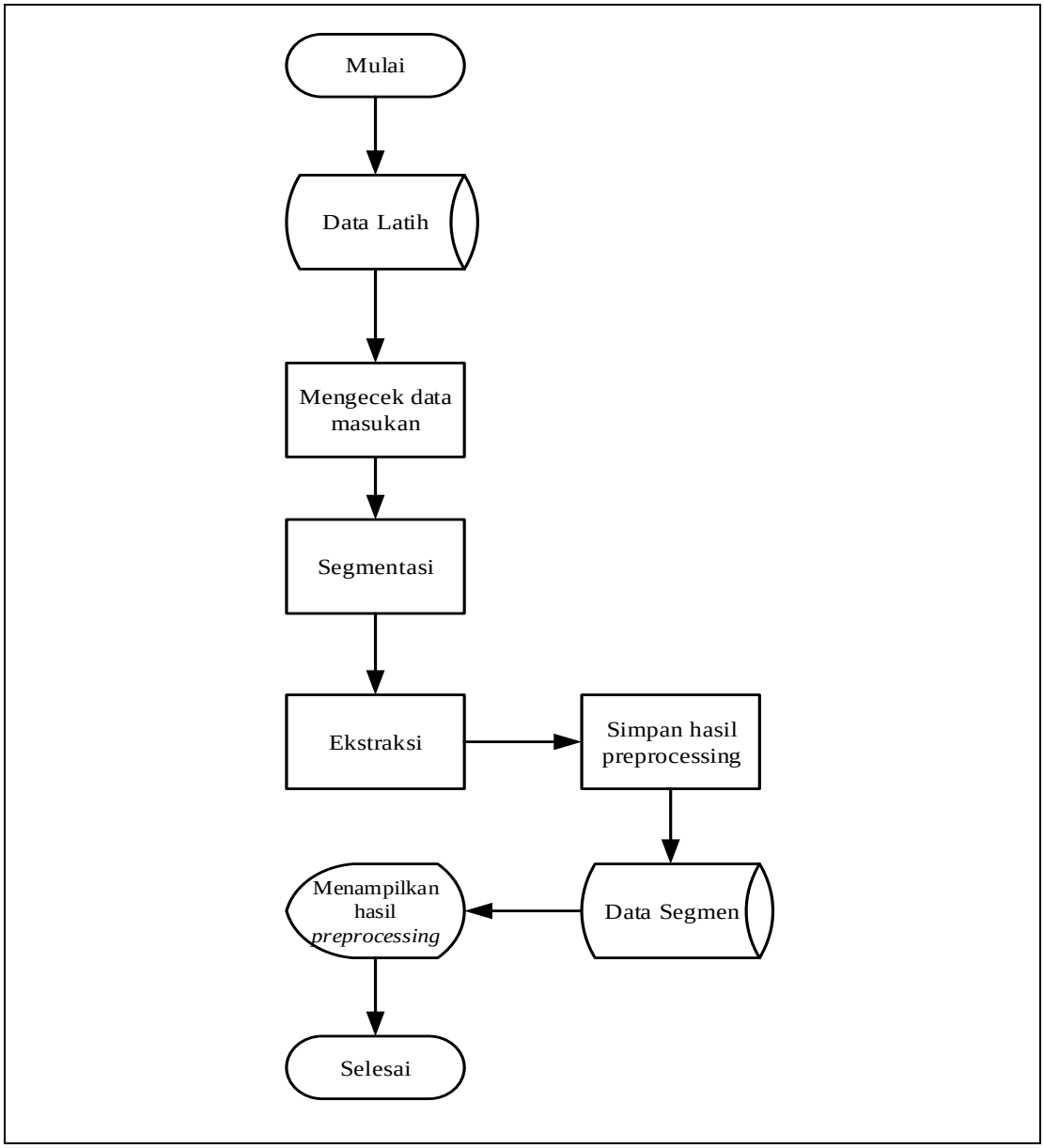
Flowchart ini merupakan prosedur yang terjadi ketika pengguna akan melakukan proses hapus data latih. Dimulai dengan memilih data latih yang akan dihapus, kemudian sistem akan menampilkan konfirmasi bahwa sistem akan dihapus atau tidak. Jika ya, maka akan dilakukan proses hapus data latih dan kemudian hasilnya akan disimpan ke dalam *database* serta ditampilkan.



Gambar 3.39 *Flowchart* Hapus Data Latih

4. *Flowchart Preprocessing*

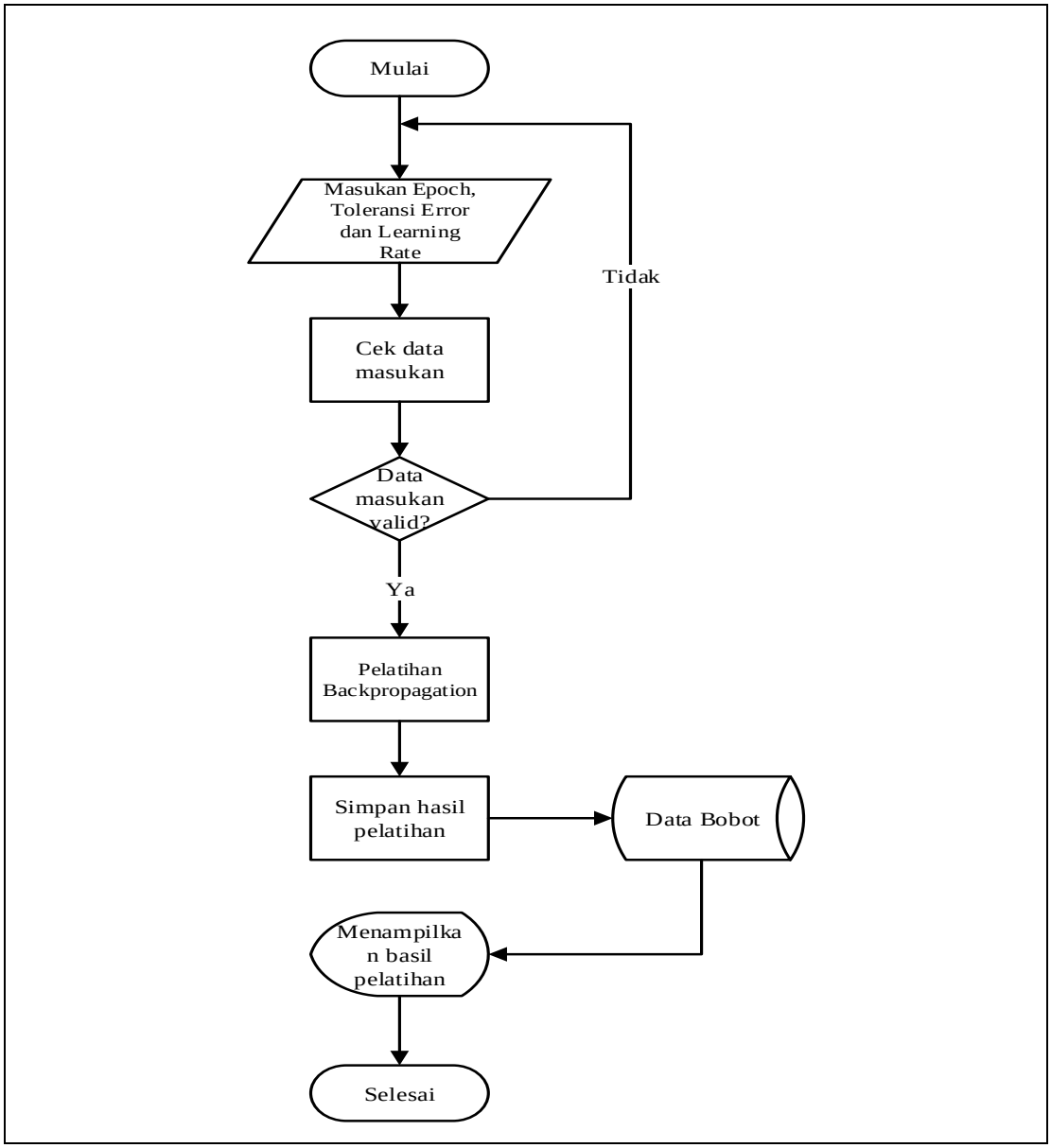
Flowchart ini merupakan prosedur yang terjadi ketika pengguna akan melakukan proses *preprocessing.* Dimulai dengan menekan tombol *preprocessing*, kemudian sistem akan mengambil data latih dari *database*. Selanjutnya sistem akan melakukan proses *preprocessing* dan kemudian hasilnya akan disimpan ke dalam *database* serta ditampilkan.



Gambar 3.40 *Flowchart Preprocessing*

5. *Flowchart* Pelatihan *Backpropagation*

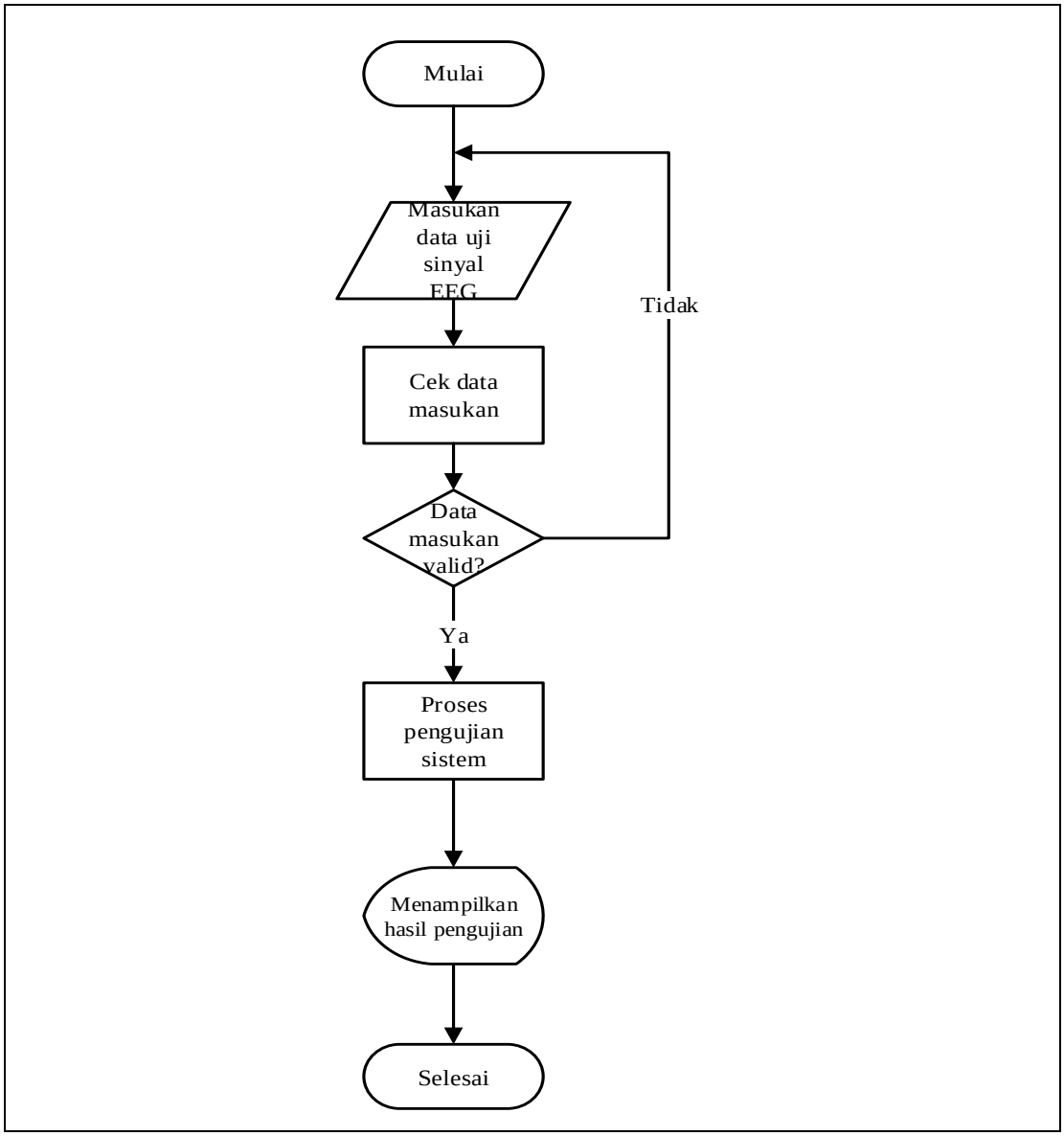
Flowchart pelatihan *backpropagation* merupakan prosedur yang terjadi ketika pengguna akan melakukan proses pelatihan *backpropagation*. Dimulai dengan memasukkan *parameter* yang digunakan pada pelatihan, kemudian sistem akan melakukan pengecekan, jika valid maka akan dilakukan proses pelatihan dan hasilnya akan disimpan ke *database* serta ditampilkan.



Gambar 3.41 *Flowchart* Pelatihan *Backpropagation*

6. *Flowchart* Pengujian *Backpropagation*

Flowchart pengujian *backpropagation* merupakan prosedur yang terjadi ketika pengguna akan melakukan proses pengujian *backpropagation*. Dimulai dengan memasukkan data, kemudian sistem akan melakukan pengecekan, jika valid maka akan dilakukan proses pengujian dan hasilnya akan disimpan ke *database* serta ditampilkan.



Gambar 3.42 *Flowchart* Pengujian *Backpropagation*

3.10.2 *Pseudocode*

Dalam perancangan sistem klasifikasi kondisi berpikir ini terdapat proses utama, yaitu segmentasi, *wavelet* dan *backpropagation*. Perancangan algoritma pada proses segmentasi adalah sebagai berikut ini.

1. Perancangan algoritma proses segmentasi

Algoritma ini digunakan untuk melakukan proses segmentasi pada sinyal EEG.

```
function SegmentasiEEG(input sinyalEEG : array[] of integer, input
: waktuSegmentasiPerDetik : integer, input samplingRate : integer)
→ array[] of integer
{Mengirim hasil segmentasi sinyal EEG.}

DEKLARASI
Segmen, temp : array[0..panjang(sinyalEEG)] of integer
i, j, waktu, fs : integer

ALGORITMA:
j ← 0
waktu ← 0
fs ← 0
for i←0 to panjang(sinyalEEG) do
    temp[j] ← sinyalEEG[i]
    j++
    if (i+1) mod (waktuSegmenPerDetik x samplingRate) == 0) then
         j ← 0
         segmen[] ← temp
    endif
endfor
return segmen
```

Algoritma berikut merupakan algoritma yang digunakan pada proses transformasi *wavelet*.

1. Perancangan algoritma konvolusi *Low-pass Filter*

Algoritma ini digunakan untuk melakukan konvolusi terhadap *low-pass*, dan *filter* yang digunakan adalah *filter daubechies* 4.

```
function KonvolusiLow(input sinyalEEG : array[] of integer) →
array[] of integer
{Mengirim hasil konvolusi terhadap low-pass filter.}

DEKLARASI
```

```
lowPassFilter     : array[0..4] of integer
hasil             : array[0..panjang(sinyalEEG)] of integer
LPF_temp          : array[0..panjang(sinyalEEG)] of integer
idx, i, i_temp, idx_temp : integer

ALGORITMA:
lowPassFilter[0] ← 0.482962913144534
lowPassFilter[1] ← 0.836516303737808
lowPassFilter[2] ← 0.224143868042013
lowPassFilter[3] ← -0.129409522551260
for i←0 to panjang(sinyalEEG) do
    hasil[] ← 0
    if i<panjang(lowPassFilter) then
        LPF_temp[i] ← lowPassFilter[i]
    else
        LPF_temp[i] ← 0
    endif
endfor
for i←0 to panjang(sinyalEEG) do
    i_temp ← i
    idx_temp ← 0
    for j←i to panjang(sinyalEEG) do
        hasil[idx] ← hasil[idx] + (sinyalEEG[i_temp] x
        LPF_temp[idx_temp])
        i_temp++
        idx_temp++
    end for
    idx++
end for
return hasil
```

2. Perancangan algoritma konvolusi *High-Pass Filter*

Algoritma ini digunakan untuk melakukan konvolusi terhadap *high-pass*, dan *filter* yang digunakan adalah *filter daubechies* 4.

```
function KonvolusiHigh(input sinyalEEG : array[] of integer) →
array[] of integer
{Mengirim hasil konvolusi terhadap high-pass filter.}

DEKLARASI
highPassFilter    : array[0..4] of integer
hasil             : array[0..panjang(sinyalEEG)] of integer
HPF_temp          : array[0..panjang(sinyalEEG)] of integer
idx, i, i_temp, idx_temp : integer

ALGORITMA:
highPassFilter[0] ← -0.129409522551260
highPassFilter[1] ← -0.224143868042013
```

```
highPassFilter[2] ← 0.836516303737808
highPassFilter[3] ← -0.482962913144534
for i←0 to panjang(sinyalEEG) do
    hasil[] ← 0
    if i<panjang(highPassFilter) then
        HPF_temp[i] ← highPassFilter[i]
    else
        HPF_temp[i] ← 0
    endif
endfor
for i←0 to panjang(sinyalEEG) do
    i_temp ← i
    idx_temp ← 0
    for j←i to panjang(sinyalEEG) do
        hasil[idx] ← hasil[idx] + (sinyalEEG[i_temp] x
        HPF_temp[idx_temp])
        i_temp++
        idx_temp++
    end for
    idx++
end for
return hasil
```

3. Perancangan algoritma *downsampling*

Algoritma ini digunakan untuk mereduksi sinyal dengan mengabaikan indeks genap pada sinyal.

```
function DownSampling(input sinyalEEG : array[] of integer, factor
: integer) → array[] of Integer
{mengirimkan sinyal yang telah direduksi.}

DEKLARASI
hasil, hasilSinyal : array[0..panjang(sinyalEEG)/factor] of integer
idx, i : integer

ALGORITMA:
hasil ← panjang(sinyalEEG)/factor
idx ← 0
for i←0 to panjang(hasil) do
    hasilSinyal[i] ← sinyalEEG[idx]
    idx ← idx + factor
endfor
return hasilSinyal
```

4. Perancangan algoritma GetTeta

Algoritma ini digunakan untuk mendapatkan gelombang teta pada sinyal masukan EEG, pada fungsi GetTeta membutuhkan fungsi lainnya yaitu konvolusiLow dan konvolusiHigh sesuai dengan *filter* yang diinginkan.

```
function GetTeta(input sinyalEEG : array[] of integer) → array[]
    of integer
{Mengirim  sinyal teta dari sinyal masukan EEG.}

DEKLARASI
A1, A1_1, AA2, AA2_2, AAA3, AAA3_3, AAAA4, AAAA4_4, AAAAA5,
AAAAA5_5, AAAAAA6, AAAAAA6_6,DAAAAAA7, DAAAAAA7_7 : array[] of
integer
i : integer

ALGORITMA:
A1 ← konvolusiLow(sinyal)
A1_1 ← downSampling(A1, 2)

AA2 ← konvolusiLow(A1_2)
AA2_2 ← downSampling(AA2, 2)

AAA3 ← konvolusiLow(AA2_2)
AAA3_3 ← downSampling(AAA3, 2)

AAAA4 ← konvolusiLow(AAA3_3)
AAAA4_2 ← downSampling(AAAA4, 2)

AAAAA5 ← konvolusiLow(AAAA4_4)
AAAAA5_5 ← downSampling(AAAAA5, 2)

DAAAAA6 ← konvolusiLow(AAAAA5_5)
DAAAAA6_6 ← downSampling(DAAAAA6, 2)

DAAAAAA7 ← konvolusiLow(AAAAAA6_6)
DAAAAAA7_7 ← downSampling(DAAAAAA7, 2)

return DAAAAAA7_7
```

5. Perancangan algoritma GetAlfa

Algoritma ini digunakan untuk mendapatkan gelombang alfa pada sinyal masukan EEG, pada fungsi GetAlfa membutuhkan fungsi lainnya yaitu konvolusiLow dan konvolusiHigh sesuai dengan *filter* yang diinginkan.

```
function GetAlfa(input sinyalEEG : array[] of Integer) → array[]
of integer
{Mengirim  sinyal alfa dari sinyal masukan EEG.}

DEKLARASI
A1,  A1_1,  AA2,  AA2_2,  AAA3,  AAA3_3,  AAAA4,  AAAA4_4,  AAAAA5,
AAAAA5_5,  DAAAAA6,  DAAAAA6_6,  ADAAAAA7,  ADAAAAA7_7,  DDAAAAA7,
DDAAAAA7_7, ADDAAAAA8, ADDAAAAA8_8, hasil : array[] of integer
i : integer

ALGORITMA:
A1 ← konvolusiLow(sinyalEEG)
A1_1 ← downSampling(A1, 2)

AA2 ← konvolusiLow(A1_1)
AA2_2 ← downSampling(AA2, 2)

AAA3 ← konvolusiLow(AA2_2)
AAA3_3 ← downSampling(AAA3, 2)

AAAA4 ← konvolusiLow(AAA3_3)
AAAA4_4 ← downSampling(AAAA4, 2)

AAAAA5 ← konvolusiHigh(AAAA4_4)
AAAAA5_5 ← downSampling(AAAAA5, 2)

DAAAAA6 ← konvolusiLow(AAAAA5_5)
DAAAAA6_6 ← downSampling(DAAAAA6, 2)

ADAAAAA7 ← konvolusiHigh(DAAAAA6_6)
ADAAAAA7_7 ← downSampling(ADAAAAA7, 2)

ADDAAAAA8 ← konvolusiLow(DDAAAAA7_7)
ADDAAAAA8_8 ← downSampling(ADDAAAAA8, 2)

for i<-0 to panjang(ADAAAAA7_7) do
    hasil[i] <- ADAAAAA7_7[i]
endfor

for i←0 panjang(ADDAAAAA8_8) do
    hasil[i] ← ADDAAAAA8_8[i]
endfor

return hasil
```

6. Perancangan algoritma GetBeta

Algoritma ini digunakan untuk mendapatkan gelombang alfa pada sinyal masukan EEG, pada fungsi GetBeta membutuhkan fungsi lainnya yaitu konvolusiLow dan konvolusiHigh sesuai dengan *filter* yang diinginkan.

```
function GetBeta(input sinyalEEG : array[] of integer) → array[]
of integer
{Mengirim  sinyal beta dari sinyal masukan EEG.}

DEKLARASI
A1, A1_1, AA2, AA2_2, AAA3, AAA3_3, AAAA4, AAAA4_4, DAAAA5,
DAAAA5_5, AAAAA5, AAAAA5_5, DAAAAA6, DAAAAA6_6, DDAAAAA7,
DDAAAAA7_7, DDDAAAAA8, DDDAAAAA8_8, hasil : array[] of integer
i : integer

ALGORITMA
A1 <- konvolusiLow(sinyalEEG)
A1_1 <- downSampling(A1, 2)

AA2 <- konvolusiLow(A1_1)
AA2_2 <- downSampling(AA2, 2)

AAA3 <- konvolusiLow(AA2_2)
AAA3_3 <- downSampling(AAA3, 2)

AAAA4 <- konvolusiLow(AAA3_3)
AAAA4_4 <- downSampling(AAAA4, 2)

DAAAA5 <- konvolusiHigh(AAAA4_4)
DAAAA5_5 <- downSampling(DAAAA5, 2)

AAAAA5 <- konvolusiHigh(AAAA4_4)
AAAAA5_5 <- downSampling(AAAAA5, 2)

DAAAAA6 <- konvolusiHigh(DAAAA5_5)
DAAAAA6_6 <- downSampling(DAAAAA6, 2)

DDAAAAA7 <- konvolusiHigh(DAAAAA6_6)
DDAAAAA7_7 <- downSampling(DDAAAAA7, 2)

DDDAAAAA8 <- konvolusiHigh(DDAAAAA7_7)
DDDAAAAA8_8 <- downSampling(DDDAAAAA8, 2)

for i<-0 to panjang(DAAAA5_5) do
    hasil[i] <- DAAAA5_5[i]
endfor

for i<-0 to panjang(DDDAAAAA8_8) do
    hasil[i] <- DDDAAAAA8_8[i]
```

```
endfor

return hasil
```

Algoritma ini adalah algoritma yang digunakan pada proses pelatihan dengan menggunakan metode *backpropagation*. Pada metode ini dilakukan beberapa proses diantaranya random nilai bobot dan bobot bias, aktivasi sigmoid, penghitungan nilai *input* pada *hidden layer*, sampai perhitungan nilai MSE yang menjadi penentu apakah pelatihan berakhir atau berlanjut untuk iterasi berikutnya dengan melihat apakah nilai MSE sudah lebih kecil dari toleransi *error*. Fungsi yang digunakan dalam pelatihan dengan *backpropagation* adalah sebagai berikut.

1. Perancangan algoritma *random* angka

Fungsi ini merupakan fungsi yang digunakan dalam mendapatkan nilai secara acak untuk bobot pada proses pelatihan.

```
function RandomAngka(input min : integer, input max : integer,
input round : integer) → real
{Mengirim nilai acak dengan rentang antara min dan max.}

DEKLARASI
randomFloat : real

ALGORITMA:
randomFloat ← min + (math.random() x (max - min))
if (round > 0) then
      randomFloat ← round(randomFloat, round)
endif
return randomFloat
```

2. Perancangan algoritma PembentukanBobot

Fungsi ini merupakan fungsi yang digunakan untuk membentuk bobot antara *input - hidden layer* dan bobot antara *hidden – output layer*.

```
function PembentukanBobot (input nilai1 : Integer, input nilai2 :
integer) → array[][] of Integer
{Mengirim nilai pembentukan bobot 2 dimensi.}

DEKLARASI
bobot : array[0..panjang(nilai1)][0..panjang(nilai2)] of integer
i, j : integer
```

```
ALGORITMA:
for i←0 to panjang(nilai1)-1 do
      for j←0 to panjang(nilai2)-1 do
            bobot[i][j] ← randomAngka(-0.5, 0.5)
      endfor
endfor

return bobot
```

3. Perancangan algoritma PembentukanBobotBias

Fungsi ini merupakan fungsi yang digunakan untuk membentuk bobot bias antara *input - hidden layer* dan bobot bias antara *hidden – output layer*.

```
function PembentukanBobotBias(input nilai : Integer) → array[] of
integer
{Mengirim hasil pembentukan bobot bias}

DEKLARASI
bobot : array[0..panjang(nilai)] of integer
i : Integer

ALGORITMA:
for i<-0 to panjang(nilai)-1 do
      bobot[i] <- RandomAngka(-0.5, 0.5)
endfor

return bobot
```

4. Perancangan algoritma AktivasiSigmoid

Fungsi ini merupakan fungsi yang digunakan untuk proses aktivasi.

```
function AktivasiSigmoid (x : real) → real
{Mengirim hasil aktivasi dengan Sigmoid Biner.}

DEKLARASI
(tidak ada)

ALGORITMA:
return (1.0/(1.0+Math.exp(-x)))
```

5. Pearancangan algoritma HitungZ_in

Fungsi ini merupakan fungsi yang digunakan untuk menghitung nilai *input* pada *hidden layer*.

```
function HitungZ_in(input X : array[] of integer, input panjangZ :
integer, input V : array[][] of real, input V0 : array[] of real)
→ array[] of real
{mengirimkan nilai Z_in dalam bentuk array.}

DEKLARASI
Z_in : : array[] of real
i, j : integer

ALGORITMA:
for j←0 to panjang(Z) do
      Z_in[j] ← V0[j]
      for i←0 to panjang(X) do
            Z_in[j] ← Z_in[j] + (X[i] x V[i][j])
      endfor
endfor
return Z_in
```

6. Perancangan algoritma HitungZ

Fungsi ini merupakan fungsi untuk menghitung nilai keluaran pada *hidden layer*.

```
function HitungZ(input Z_in : array[] of real) → array[] of real
{mengirimkan nilai Z dalam bentuk array.}

DEKLARASI
Z : : array[] of real
j : integer

ALGORITMA:
for j←0 to panjang(Z) do
      Z[j] ← aktivasiSigmoid(Z_in[j])
endfor
return Z
```

7. Perancangan algoritma HitungY_in

Fungsi ini merupakan fungsi yang digunakan untuk menghitung nilai *input* pada *output layer*.

```
function HitungY_in(input Z : array[] of integer, input panjangY :
integer, input W : array[][] of real, input W0 : array[] of real)
→ array[] of real
{mengirimkan nilai Y_in dalam berntuk array.}

DEKLARASI
Y_in: array[] of real
j, k : Integer

ALGORITMA:
for k←0 to panjang(Y)-1 do
      Y_in[k] ← W0[k]
      for j←0 to panjang(Z)-1 do
            Y_in[k] ← Y_in[k] + (Z[i] x W[j][k])
      endfor
endfor
return Y_in
```

8. Perancangan algoritma HitungY

Fungsi ini merupakan fungsi yang digunakan untuk menghitung nilai keluaran pada *output layer*.

```
function HitungY(input Y_in : array[] of real) → array[] of real
{mengirimkan nilai Y dalam bentuk array.}

DEKLARASI
Y : array[] of real
k : integer

ALGORITMA
for k←0 to panjang(Y_in)-1 do
      Y[k] ← AktivasiSigmoid(Y_in[k])
endfor
return Y
```

9. Perancangan algoritma HitungThoY

Fungsi ini merupakan fungsi untuk menghitung nilai kesalahan di *output layer*.

```
function HitungThoY(input Y : array[] of real, input T : array[]
of integer)→ array[] of real
{mengirimkan nilai ThoY dalam bentuk array.}

DEKLARASI
ThoY : array[] of real
k : integer

ALGORITMA:
for k←0 to panjang(Y) do
      ThoY[k] ← (T[k]-Y[k]) x (Y[k]) x (1-Y[k])
endfor
return ThoY
```

10. Perancangan algoritma HitungDW

Fungsi ini merupakan fungsi untuk menghitung nilai untuk koreksi bobot antara *hidden – output layer*.

```
function HitungDW(input learning : real, input ThoY : array[] of
real, input Y : array[] of real, input Z : array[] of real)→
array[][] of real
{mengirimkan nilai DW dalam bentuk array dua dimensi.}

DEKLARASI
DW : array[][] of real
j, k : integer

ALGORITMA:
for j←0 to panjang(Z)-1 do
      for k←0 to panjang(Y)-1 do
            DW[j][k] ← learning x Z[j] x ThoY[k]
      endfor
endfor
return DW
```

11. Perancangan algoritma HitungDW0

Fungsi ini merupakan fungsi untuk menghitung nilai koreksi bobot bias antara *hidden – output layer*.

```
function HitungDW0(input learning : real, input Y : array[] of
real, input ThoY : array[] of real)→ array[] of real

{mengirimkan nilai DW0 dalam bentuk array.}

DEKLARASI
DW0 : array[] of real
k : integer

ALGORITMA
for k←0 to panjang(Y) do
      DW0[k] ← learning x ThoY[k]
endfor
return DW0
```

12. Perancangan algoritma HitungThoZ_in

Fungsi ini merupakan fungsi untuk menghitung nilai kesalahan *input* pada *hidden layer*.

```
function HitungThoZ_in(input Y : array[] of real, input Z :
array[] of real, input ThoY : array[] of real, input W : array[][]
of real)→ array[] of real
{mengirimkan nilai ThoZ_in dalam bentuk array.}

DEKLARASI
ThoZ_in : array[] of real
j, k : integer

ALGORITMA:
for j←0 to panjang(Z) do
      ThoZ_in[j] ← 0
      for k←0 to panjang(Y) do
            ThoZ_in[j] ← ThoZ_in[j] + (ThoY[k] x W[j][k])
      endfor
endfor
return ThoZ_in
```

13. Perancangan algoritma HitungThoZ

Fungsi ini merupakan fungsi untuk menghitung nilai kesalahan pada *hidden layer*.

```
function HitungThoZ(input Z_in : array[] of real, input ThoZ_in : array[] of real)→ array[] of real
{mengirimkan nilai ThoZ dalam bentuk array.}

DEKLARASI
ThoZ : array[] of real
j : integer

ALGORITMA:
for j←0 to panjang(Z) do
    ThoZ[j] ← (ThoZ_in[j]) x (Z[j]) x (1-Z[j])
endfor
return ThoZ
```

14. Perancangan algoritma HitungDW

Fungsi ini digunakan untuk menghitung koreksi bobot antara *input – hidden layer*.

```
function HitungDV(input learning : real, input X : array[] of integer , input Z : array[] of real , input ThoZ : array[] of real)→ array[][] of real
{mengirimkan nilai DV dalam bentuk array dua dimensi.}

DEKLARASI
DV : array[][] of real
i, j : Integer

ALGORITMA:
for i←0 to panjang(X) do
      for j←0 to panjang(Z) do
            DW[i][j] ← learning x X[i] x ThoZ[j]
      endfor
endfor
return DV
```

15. Perancangan algoritma HitungDV0

Fungsi ini digunakan untuk menghitung koreksi bobot bias antara *input – hidden layer*.

```
function HitungDV0(input learning : real, input Z : array[] of
real , input ThoZ : array[] of real)→ array[] of real
{mengirimkan nilai DV0 dalam bentuk array.}

DEKLARASI
DV0 : array[] of real
j : integer

ALGORIMA:
for j←0 to panjang(Z) do
      DV0[j] ← learning x ThoY[j]
endfor
return DV0
```

16. Perancangan algoritma HitungMSE

Fungsi ini digunakan untuk menghitung rata-rata error pada setiap *iterasi*.

```
function HitungMSE(input Y : array[] of real , input T : array[]
of real)→ array[] of real
{mengirimkan nilai MSE_data.}

DEKLARASI
MSE_data : real
k : integer

ALGORITMA:
MSE_data ← 0

for k←0 to panjang(Y) do
      MSE_data ← MSE_data + ((T[k]-Y[k]) x (T[k]-Y[k]))
end for

MSE_data ← MSE_data / panjang(Y)
return MSE_data
```

17. Perancangan algoritma PerbaharuiBobot

Prosesdur ini digunakan untuk memperbaharui semua bobot lama dengan bobot baru.

```
procedure PerbaharuiBobot(input X : array[] of real, input Z : array[] of real, input Y : array[] of real, input DV : array[][] of real, input DV0 : array[] of real, input DW : array[][] of real, input DW0 : array[] of real, input/output V : array[][] of real, input/output V0 : array[] of real, input/output W : array[][] of real, input/output W0 : array[] of real)
{I.S : X[], Z[], Y[], DV[][], DV0[], DW[][], DW0[], V[][], V0[], W[][], W0[] telah diketahui}
{F.S : V[][], V0[], W[][], W0[] di perbaharui dengan menjumlahkannya terhadap DV[][], DV0[], DW[][], DW0[]}

DEKLARASI
i, j, k : Integer

ALGORITMA:
for i←0 to panjang(X) do
for j←0 to panjang(Z) do
                  V[i][j] ← V[i][j] + DV[i][j]
      endfor
endfor

for j←0 to panjang(Z) do
V0[j] ← V0[j] + DV0[j]
endfor

for j←0 to panjang(Z) do
      for k←0 to panjang(Y) do
            W[j][k] ← W[j][k] + DW[j][k]
      endfor
endfor

for k←0 to panjang(Y) do
W0[k] ← W0[k] + DW0[k]
endfor
```

18. Perancangan algoritma PelatihanBP

Fungsi ini merupakan fungsi utama yang digunakan pada proses pelatihan JST menggunakan algoritma *backpropagation*.

```
procedure PelatihanBP()
{I.S : Semua data diketahui}
{F.S : V[][], V0[], W[][] dan W0[] disimpan ke dalam database}

DEKLARASI
idx_epoch, idx_data : integer

AGORITMA:
for idx_epoch←0 to epoch do
      for idx_data←0 to panjang(data) do
            HitungZ_in()
            HitungZ()
            HitungY_in()
            HitungY()
            HitungThoY()
            HitungDW()
            HitungDW0()
            HitungThoZ_in()
            HitungThoZ()
            HitungDV()
            HitungDV0()
            UpdateBobot()
            HitungMSE_data()
            MSE ← MSE + MSE_data
      endfor

      MSE ← MSE / panjang(data)
      if MSE ≤ maxError then
            break
      endif
endfor

MSE_data ← MSE_data / panjang(Y)
```

Algoritma ini digunakan untuk proses pengujian dengan *backpropagation*. Pada fungsi ini digunakan juga beberapa fungsi yang digunakan pada saat pelatihan *backpropagation* tanpa melakukan iterasi dengan *epoch* tertentu dan pengecekan MSE, termasuk perubahan bobot-bobotnya.

1. Perancangan algoritma PengujianBP

Pada perancangan algoritma PengujianBP hanya melakukan sampai penghitungan nilai pada *neuron* keluaran, lalu membandingkan sesuai kelas yang sudah ditentukan sebelumnya.

```
procedure PengujianBP()
{I.S : Semua data diketahui}
{F.S : Menampilkan hasil klasifikasi}

DEKLARASI
idx_epoch, idx_data : integer

ALGORITMA:
HitungZ_in()
HitungZ()
HitungY_in()
HitungY()

if Y[0] > Y[1] then
      output ("Rileks")
else
      output ("Non-Rileks")
endif
```