BAB III
RANCANGAN SISTEM

3.1 Spesifikasi Kulkas Pintar

Kulkas pintar yang dirancang memiliki spesifikasi sebagai berikut:

a. Memiliki 8 buah sensor cahaya (*photo resistor*) untuk memantau jumlah telur dalam kulkas.
b. Menggunakan satu buah sensor suhu LM35 untuk memantau suhu internal kulkas.
c. Menggunakan protokol komunikasi ZigBee untuk melakukan komunikasi dengan *gateway*.
d. Pengecekan jumlah telur dan suhu kulkas dilakukan hanya pada saat pintu kulkas dibuka.
e. Akses informasi dari kulkas dilakukan melalui aplikasi Android yang menggunakan protokol XML-RPC untuk berkomunikasi dengan *server*.

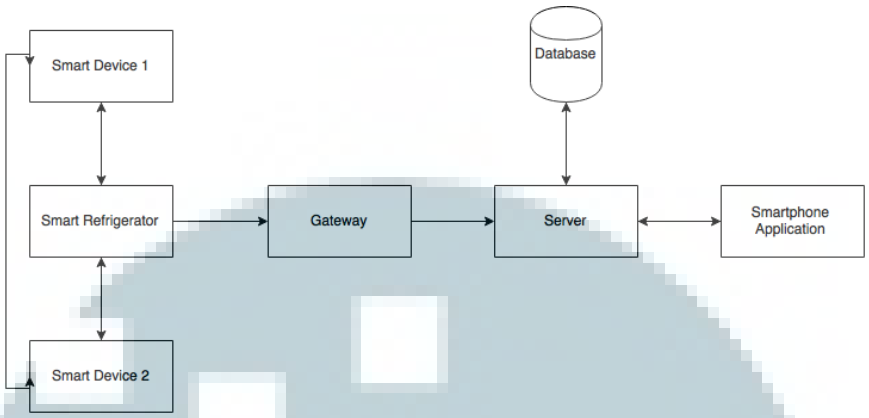
3.2 Rancangan Sistem Keseluruhan

Gambar 3.1 merupakan diagram blok rancangan sistem kulkas pintar secara keseluruhan. Pada sistem ini, kulkas pintar dirancang sedemikian rupa agar dapat berkomunikasi dengan perangkat pintar lainnya yang berada pada jaringan yang sama. Data yang dihasilkan oleh perangkat kemudian dikirimkan ke *gateway*.

Data dari *device* kemudian diolah oleh *gateway* untuk diteruskan ke *server* yang berada di *cloud* dan disimpan ke *database*. Selain bertugas mengolah data dari *device*, *gateway* juga bertugas untuk membuat sebuah PAN, agar dapat menghubungkan semua *device* yang berada di suatu tempat.

Data yang telah diolah *gateway* selanjutnya dikirimkan ke *server* menggunakan XML-RPC. *Server* kemudian mengeksekusi *method* yang sesuai dengan yang diminta oleh *gateway*. Data yang dikirimkan oleh *gateway* kemudian akan disimpan ke dalam suatu *database* agar dapat diakses oleh suatu aplikasi berbasis Android.

Pengguna dapat mengakses informasi mengenai perangkat yang diinginkannya melalui suatu aplikasi berbasis Android. Aplikasi ini berkomunikasi dengan *server* menggunakan protokol XML-RPC. Agar dapat digunakan, pengguna harus mendaftarkan dirinya terlebih dahulu ke *server*. Setelah terdaftar, pengguna dapat menggunakan *username* dan *password*-nya untuk masuk ke aplikasi. Setelah berhasil masuk, pengguna kemudian dapat mengakses informasi yang telah dikirimkan oleh perangkat pintar.



Gambar 3.1 Diagram blok sistem keseluruhan

3.3 Rancangan Perangkat Keras Sistem Sensor

Komponen-komponen yang digunakan untuk membangun kulkas pintar adalah sebagai berikut:

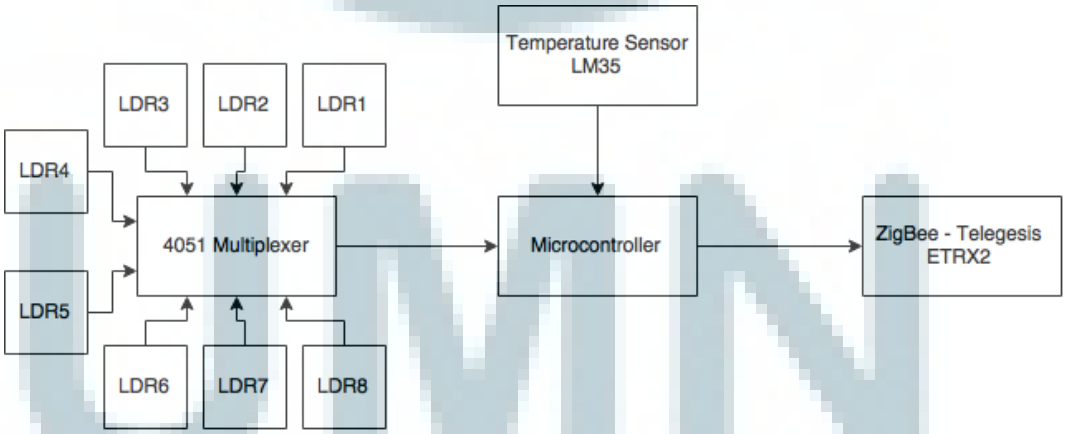
a. Satu buah sensor suhu LM35
b. CENTER 314 *Humidity Temperature Meter* sebagai pembanding bacaan sensor suhu
c. Delapan buah sensor cahaya *photo resistor*
d. Multiplekser 4051
e. Mikrokontroler Atmega328 beserta *board* Arduino Uno Rev. 3
f. Modul ZigBee Telegesis ETRX2. Satu sebagai *coordinator* dan yang satu sebagai *end device*. *Coordinator* dihubungkan dengan *gateway*.

Gambar 3.2 merupakan diagram blok dari sistem sensor pada kulkas pintar. Terdapat delapan buah *light dependant resistor* yang dihubungkan ke multiplekser 4051 untuk mendeteksi keberadaan telur dengan cara membaca gelap terangnya cahaya. Selain itu, juga terdapat sebuah sensor

suhu LM35 yang digunakan untuk mengetahui suhu di dalam kulkas. Kedua jenis sensor ini dihubungkan ke mikrokontroler.

Mikrokontroler bertugas untuk menerima bacaan dari setiap sensor dan meneruskannya kepada server. Mikrokontroler juga menyimpan *device* ID yang akan menjadi pengenal kulkas pintar pada saat berkomunikasi dengan *gateway* maupun dengan perangkat lainnya. Alamat EUI64 dari *gateway* dan perangkat pintar yang berkomunikasi dengan kulkas pintar juga disimpan oleh mikrokontroler.

Agar kulkas pintar dapat berkomunikasi dengan perangkat lainnya, maka digunakan sebuah modul ZigBee Telegesis ETRX2 yang dihubungkan ke mikrokontroler. Modul ini berkomunikasi dengan modul ZigBee pada *gateway* maupun dengan modul pada perangkat pintar lainnya.



Gambar 3.2 Diagram blok perangkat keras kulkas pintar

3.4 Protokol Komunikasi Lokal

3.3.1 Protokol Umum

3.3.1.1 Tipe Device

Terdiri dari dua huruf kapital berpasangan. *Gateway* menggunakan tipe device ZZ. Kulkas menggunakan tipe *device* RF.

3.3.1.2 Nomor Device

Untuk menentukan nomor seri suatu *device* yang sama. Terdiri dari 3 digit heksadesimal (000 – FFF). Misalnya lampu 1 000, lampu 2 00A, lampu 3 00F.

3.3.1.3 Device ID

Tipe dan Nomor Device akan digabung sebagai suatu *identifier* bagi suatu *device*. Misalnya kulkas adalah RF 000, lampu 1 AB 000, lampu 3 AB 00F.

3.3.2 Protokol Komunikasi Lokal dengan Zigbee

3.3.2.1. Mendaftarkan diri ke *database*

a. *Gateway* mengirimkan *introduction_message* ke semua *device*

   Format pesan yang dikirimkan:

```
at+annce<CR>
```

b. Menerima *introduction_message* dari *Gateway*

   Format pesan yang diterima:

```
<CR><LF>FFD:[EUI-64 gateway],[Node ID gateway]<CR><LF>
```

   Contoh:

<CR><LF>FFD:000D6F0002382C14,0000<CR><LF>

Catatan:

*jika status pada *device* sudah bernilai 1 (sudah ditambahkan), maka pesan yang diterima akan diabaikan

**device* baru akan mengubah status menjadi 1 dan lanjut ke langkah berikutnya

c. Mengirimkan alamat EUI-64-nya dan *device* ID-nya ke *Gateway*

Format pesan yang dikirim:

```
at+ucast:[EUI-64 gateway]=DeviceID#[Device ID]#<CR>
```

Contoh:

at+ucast: 000D6F0002382C14=DeviceID#RF 001#<CR>

d. *Gateway* menerima EUI-64 dan *device* ID dari *device* baru

Format pesan yang diterima:

```
<CR><LF>UCAST:[EUI-64 device],06=DeviceID#[Device ID]#<CR><LF>
```

Contoh:

<CR><LF>UCAST: 000D6F0002382BD5,10=DeviceID#RF 001#<CR><LF>

Catatan:

*data yang diterima dimasukkan ke *database*

e. *Gateway* mengirimkan *device* ID *Gateway* ke *device* baru

Format pesan yang dikirim:

```
at+ucast:[EUI-64 device]=GateID#[Device ID gateway]#<CR>
```

Contoh:

at+ucast: 000D6F0002382BD5=GateID#ZZ 001#<CR>

f. *Device* baru mendapatkan GateID dari *gateway*

Format pesan yang diterima:

```
<CR><LF>UCAST:[EUI-64 gateway],0E=GateID#[Device ID gateway]#<CR><LF>
```

Contoh:

<CR><LF> UCAST:000D6F0002382C14,0E=GateID#ZZ

001#<CR><LF>

3.3.2.2. *Liveness update*

a. *Gateway* mengirimkan *liveness update* ke *device* untuk mengetahui apakah *device* masih hidup

Format pengiriman *liveness update*:

```
at+ucast:[EUI-64 device yang ingin diketahui liveness-nya]=<CR>
```

Contoh:

at+ucast: 000D6F0002382BD5=<CR>

Jika berhasil, contoh balasan dari modul:

<CR><LF>SEQ:AD<CR><LF>

<CR><LF>OK<CR><LF>

<CR><LF>ACK:AD<CR><LF>

Jika tidak berhasil, contoh balasan dari modul:

<CR><LF>SEQ:AB<CR><LF>

<CR><LF>OK<CR><LF>

<CR><LF>NACK:AB<CR><LF>

*gateway tidak mendapatkan balasan dari device. Untuk mengetahui, dapat dilihat dari ACK atau NACK.

b. *Device* menerima *liveness update* dari *gateway*

Format data yang diterima:

```
<CR><LF>UCAST:[EUI-64 gateway],01=<CR><LF>
```

Contoh format data yang diterima:

UCAST:000D6F0002382C14,01=<CR>

Catatan:

*Data yang diterima tidak diproses oleh *device*

*"*Gateway*" dapat diganti dengan "*device* lain"

3.3.2.3. Mengirimkan *update* ke *Gateway*

a. *Update_packet* ditransimisikan oleh *device* ke *gateway*

Tabel 3.1 Format update_packet kulkas

Fungsi	Data_Type	Format dan Contoh Update_Packet
Update Jumlah Telur	stok-suhu	Format: at+ucast:[EUI-64 *gateway*]=[Device ID]#stok-suhu#[stok dalam desimal-suhu dalam desimal]#<CR> Contoh: at+ucast: 000D6F0002382C14=RF 001#stok-suhu#7-9#<CR>

b. *Gateway* menerima *update_packet*

Format yang diterima:

```
<CR><LF>UCAST:[EUI-64 device],[DataLength]=[Device
ID]#[Data_Type]#[value]#<CR><LF>
```

Contoh:

<CR><LF>UCAST:000D6F0002382BD5,12=RF 001#stok-suhu#7-9#<CR><LF>

c. *Gateway* membalas dengan ACK

Format ACK:

```
at+ucast:[EUI-64 device]=ACK#[Data_Type]#<CR>
```

Contoh:
at+ucast: 000D6F0002382BD5=ACK#temperature#<CR>

d. *Device menerima* ACK

Format ACK yang diterima:

```
<CR><LF>UCAST:[EUI-64
gateway],[DataLength]=ACK#[Data_Type]#<CR><LF>
```

Contoh:
<CR><LF>UCAST:000D6F0002382C14,0C=ACK#stok-suhu#<CR><LF>

3.3.2.4. *Request* data dari *device* A ke *device* B

a. *Device* A langsung mengirimkan *request_packet* ke *device* B untuk mengambil data yang ada di *device* B.

i. *Device* B: *Gateway*

Tabel 3.2 Format request_packet ZigBee gateway

Fungsi	Data_Type	Format dan Contoh Request_Packet
Meminta EUI-64 kulkas	*Address*-RF	Format:

		AT+UCAST:[EUI-64 *gateway*]=[Device ID *gateway*]#address-RF#[EUI-64 kulkas]<CR> Contoh: AT+UCAST:[000D6F0002382C14]=ZZ 001#address-RF#<CR>
Meminta EUI-64 *device* lain	*address*-[Tipe *Device*]	Format: AT+UCAST:[EUI-64 tempat makanan]=[*Device ID gateway*]#address-[Tipe *Device*]# <CR> Contoh: at+ucast:000D6F0002382C14=ZZ 001#address-RF#<CR>

ii. *Device* B: Kulkas

Tabel 3.3 Format request_packet ZigBee kulkas

Fungsi	Data_Type	Format dan Contoh Request_Packet
Meminta nilai stok dan suhu kulkas	stok-suhu	Format: AT+UCAST:[EUI-64 kulkas]=stok-suhu #<CR> Contoh: AT+UCAST:000D6F0002383BD5= stok-suhu #<CR>

b. *Device* B menerima *request_packet* dari *device* A

<CR><LF>UCAST:[EUI-64 device A],[DataLength]=[Data_Type]#<CR><LF>

Device B menyimpan sementara EUI-64 *Device* A untuk pengiriman update_packet.

c. *Device* B mengirimkan *update_packet* ke *device* A sesuai dengan data yang diminta.

Langkah c-f sama dengan bagian mengirimkan *Update_packet* ke *gateway* langkah a-d dengan menganggap *gateway* (di bagian tersebut) sebagai *device* A.

Update_Packet jika *device* B adalah *gateway*:

Tabel 3.4 Format update_packet ZigBee gateway

Fungsi	Data_Type	Format dan Contoh Update_Packet
Memberikan EUI-64 kulkas	address-RF	Format: AT+UCAST:[EUI-64 *device* A]=[*Device ID gateway*]address-RF#[EUI-64 kulkas yang dicari alamatnya]#<CR> Contoh: AT+UCAST:000D6F0002382C14=ZZ001#address-RF#000ABC0001232321<CR> Jika, EUI-64 yang dicari tidak terdaftar di *database*, maka nilai

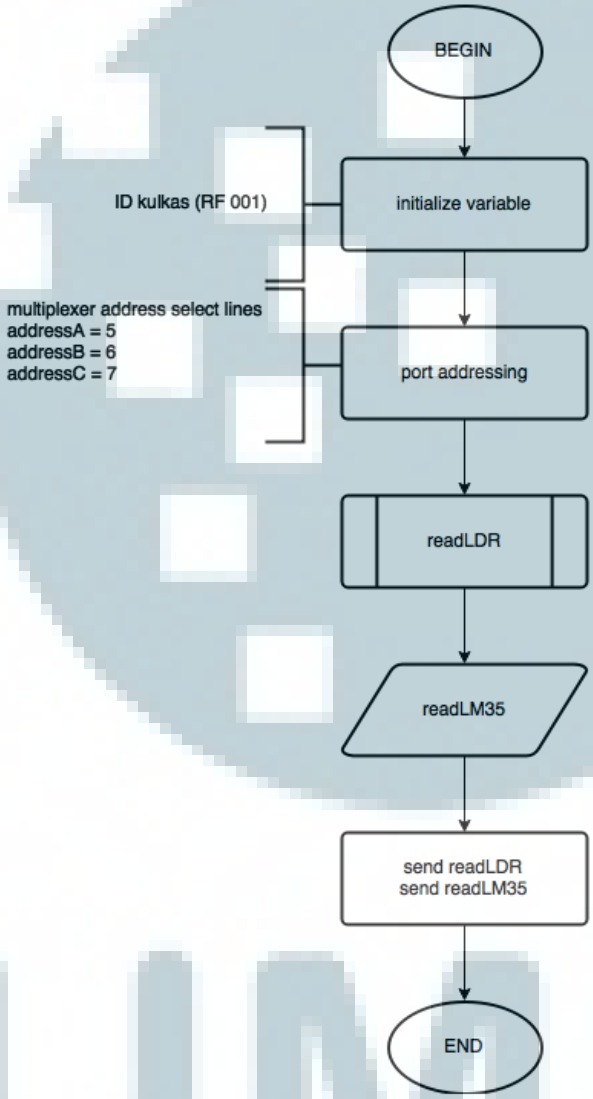
		tersebut akan diisi dengan “0” (tanpa tanda petik)
Meminta EUI-64 *device* lain	address-[Tipe *Device*]	Format: AT+UCAST:[EUI-64 *device* A]=[*Device* ID *gateway*]#address-[Tipe *Device*]# [EUI-64 *device* yang dicari alamatnya]<CR> Contoh: AT+UCAST:000D6F0002382C14=RF001#address-AA#000ABC0E01232CBA<CR> Jika, EUI-64 yang dicari tidak terdaftar di *database*, maka nilai tersebut akan diisi dengan “0” (tanpa tanda petik)

d. *Device* A menerima *update_packet* dari *device* B
e. *Device* A membalas dengan ACK ke *device* B
f. *Device* B menerima pesan ACK

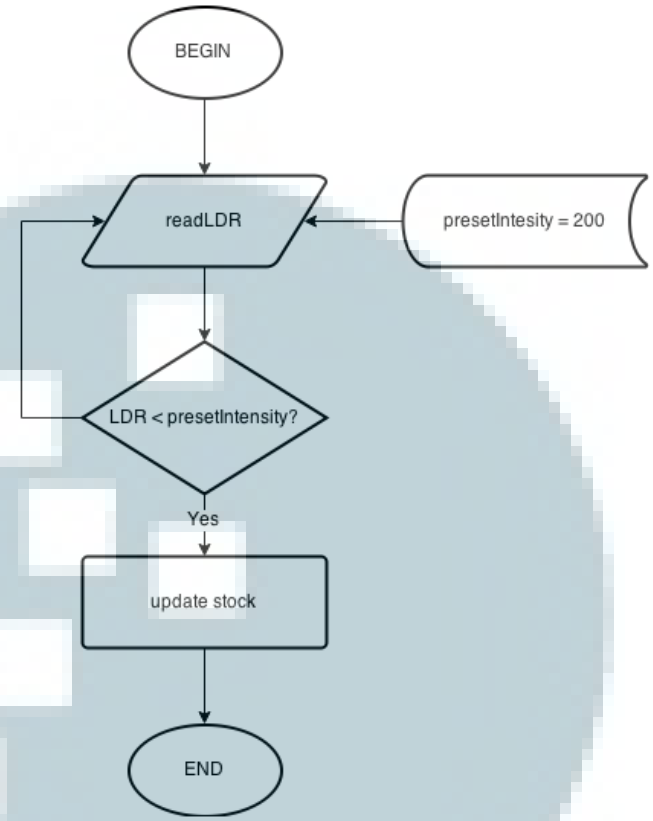
3.5 Rancangan Perangkat Lunak Kulkas

Program dibuat menggunakan bahasa C. Komunikasi *serial* dengan modul ZigBee, menggunakan *baud rate* 9600 bps. Fungsi readLDR pada Gambar 3.4 merupakan proses tersendiri yang diagram alirnya dapat dilihat

pada Gambar 3.5. Fungsi tersebut masing-masing mengirimkan suatu nilai yang berikutnya dikirimkan oleh *gateway* ke *server*. *Update* stok dan suhu dikirimkan ke *gateway* setiap kali pintu kulkas terbuka.



Gambar 3.3 Diagram alir program utama



Gambar 3.4 Diagram alir sensor cahaya

3.6 Rancangan Komunikasi antara User Application dan Server

Server dan *user* berkomunikasi dengan protokol XML-RPC. Berikut adalah fungsi-fungsi dalam *server* yang dapat dipanggil oleh *user*:

1. **addUsername:** fungsi untuk menambah *username* baru pada *database*. Aplikasi mengirimkan *XML-RPC* dengan *methodName = addUsername* dan parameter dalam sebuah *struct* dengan perincian seperti pada tabel 3.5.

Tabel 3.5 Struktur struct *fungsi addUsername* request

Key	**Tipe Data**	**Keterangan**

USERNAME	String 20	*Username* baru yang akan ditambahkan ke *database* yang sebelumnya dicek oleh *server* apakah *username* tersebut sudah ada atau belum dengan findUsernamе.
PASSWORD	String 64	*Password* dari *username* yang ditambahkan, yang telah di-*hash* dengan SHA3-256.

Fungsi ini mengembalikan nilai ke aplikasi dalam sebuah *struct* dengan nilai seperti pada Tabel 3.6.

Tabel 3.6 Struktur struct *fungsi addUsername* response

Key	**Tipe Data**	**Keterangan**
RESPONSECODE	String	'00' : *username* baru berhasil ditambahkan. '01' : gagal.
MESSAGE	String	Berisi "*Succeed*" jika berhasil. Berisi pesan *error* jika gagal.

Berikut adalah contoh pesan *request* yang dikirimkan dari aplikasi.

```
<?xml version="1.0"?>
<methodCall>
<methodName>addUsername</methodName>
<params>
  <param>
    <value>
      <struct>
        <member>
```

```
          <name>USERNAME</name>
          <value><string>tralala</string></value>
        </member>
        <member>
          <name>PASSWORD</name>
          <value><string>086abe9e75baba382d84c1a
            eee6aacec9874aac7799924e286ea000362d
            4db82</string></value>
        </member>
      </struct>
    </value>
  </param>
</params>
</methodCall>
```

Berikut adalah contoh *response* yang diberikan *server*.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>RESPONSECODE</name>
            <value><string>00</string></value>
          </member>
          <member>
            <name>MESSAGE</name>
            <value><string>Succeed</string></value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

2. **findUsername**: fungsi untuk mengetahui apakah *username* yang bersangkutan ada di dalam *database*. Fungsi mengembalikan *username* dan *password*-nya jika ada di *database*. Aplikasi mengirimkan XML-RPC dengan methodName = findUsername dan parameter dalam sebuah *struct* dengan perincian seperti pada tabel 3.7.

Tabel 3.7 Struktur struct fungsi findUsername request

Key	**Tipe Data**	**Keterangan**
USERNAME	String 20	*Username* yang ingin diketahui keberadaannya atau ingin diketahui *password*-nya jika ada di *database*.

Fungsi ini mengembalikan nilai ke aplikasi, dengan sebuah *struct* dengan nilai seperti pada Tabel 3.8.

Tabel 3.8 Struktur struct *fungsi findUsername* response

Key	**Tipe Data**	**Keterangan**
RESPONSECODE	String	'00' : *username* tidak ada di *database*. '01' : *username* ada di *database*.
MESSAGE	String	Berisi *password* jika *username* ada di *database*. Berisi "*username not exist*" jika *username* tidak ada di *database*. Password yang dikirimkan berupa hasil *hash* dengan SHA3-256.

Berikut adalah contoh format pesan *request* yang dikirimkan ke *server*.

```
<?xml version="1.0"?>
<methodCall>
<methodName>findUsername</methodName>
  <params>
    <param>
      <value>
        <struct>
```

```
          <member>
            <name>USERNAME</name>
            <value><string>tralala</string></value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

Berikut adalah contoh *response* yang diberikan *server*.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>RESPONSECODE</name>
            <value><string>01</string></value>
          </member>
          <member>
            <name>MESSAGE</name>
            <value><string>086abe9e75baba382d84c1a
              eee6aacec9874aac7799924e286ea000362d
              4db82</string>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

3. **rf.changeLowerBound:** fungsi ini dipakai untuk mengirimkan perubahan batas bawah jumlah telur yang diinginkan pengguna ke *server*. Aplikasi mengirimkan XML-RPC dengan methodName = rf.changeLowerBound dan parameter dalam sebuah *struct* dengan perincian seperti pada tabel 3.9.

Tabel 3.9 Struktur struct *fungsi rf.changeLowerBound* request

Key	**Tipe Data**	**Keterangan**

USERNAME	String 20	*Username* yang akan mengambil detail kulkas.
PASSWORD	String 64	*Password* yang telah di-*hash* dengan SHA3-256 untuk autentikasi.
DEVICE ID	String	Device ID tujuan yang batas bawahnya ingin diganti.
LOWERBOUND	Int	Batas bawah yang ingin untuk disimpan ke dalam *database*.

Fungsi ini mengembalikan nilai ke aplikasi dalam sebuah *array of struct* dengan nilai *struct* seperti pada tabel 3.10.

Tabel 3.10 Struktur struct *(dalam* Array*) fungsi rf.changeLowerBound* response

Key	**Tipe Data**	**Keterangan**
RESPONSECODE	String	'00' : *username* tidak ada di *database*. '01' : *username* ada di *database*.
MESSAGE	String	Berisi *"Succeed"* jika *username* ada di *database*. Berisi "*error*" jika tidak berhasil melakukan *update*.

Berikut adalah contoh *request* yang dikirimkan oleh aplikasi.

```
<?xml version="1.0"?>
<methodCall>
<methodName>rf.changeRefrigeratorLowerBound</methodName>
<params>
  <param>
    <value>
      <struct>
```

```
        <member>
          <name>USERNAME</name>
          <value><string>tralala</string></value>
        </member>
        <member>
          <name>PASSWORD</name>
          <value><string>086abe9e75baba382d84c1a
            eee6aacec9874aac7799924e286ea000362d
            4db82</string></value>
        </member>
        <member>
          <name>DEVICEID</name>
          <value>RF 001</value>
        </member>
        <member>
          <name>LOWERBOUND</name>
          <value>5</value>
        </member>
      </struct>
    </value>
  </param>
</params>
</methodCall>
```

Berikut adalah contoh *response* yang diberikan *server*.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>RESPONSECODE</name>
            <value><string>01</string></value>
          </member>
          <member>
            <name>MESSAGE</name>
            <value><string>success</string></value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

4. **rf.getRefrigeratorLowerBound:** fungsi ini dipakai untuk mendapatkan batas bawah jumlah telur ke *server*. Aplikasi mengirimkan XML-RPC dengan methodName =

rf.getRefrigeratorLowerBound dan parameter dalam sebuah *struct* dengan perincian seperti pada tabel 3.11.

Tabel 3.11 Struktur struct *fungsi rf.getRefrigeratorLowerBound* request

Key	**Tipe Data**	**Keterangan**
USERNAME	String 20	*Username* yang akan mengambil batas bawah kulkas.
PASSWORD	String 64	*Password* yang telah di-*hash* dengan SHA3-256 untuk autentikasi.

Fungsi ini mengembalikan nilai ke aplikasi dalam sebuah *array of struct* dengan nilai *struct* seperti pada Tabel 3.12.

Tabel 3.12 Struktur struct *(dalam* Array*) fungsi rf.getCompleteRefrigerator* response

Key	**Tipe Data**	**Keterangan**
DEVICEID	String	Device ID yang batas bawahnya ingin diketahui.
LOWERBOUND	Int	Batas bawah *device* yang diinginkan.

Berikut adalah contoh *request* yang dikirimkan dari aplikasi.

```
<?xml version="1.0"?>
<methodCall>
<methodName>rf.getCompleteRefrigerator</methodName>
<params>
  <param>
    <value>
      <struct>
        <member>
          <name>USERNAME</name>
          <value><string>tralala</string></value>
        </member>
        <member>
          <name>PASSWORD</name>
```

```
          <value><string>086abe9e75baba382d84c1a
            eee6aacec9874aac7799924e286ea000362d
            4db82</string></value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

Berikut adalah contoh *response* yang diberikan server.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value>
              <struct>
                <member>
                  <name>DEVICEID</name>
                  <value><string>RF 001</string></value>
                </member>
                <member>
                  <name>LOWERBOUND</name>
                  <value><int>5</int></value>
                </member>
              </struct>
            </value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodResponse>eway.
```

5. **rf.getCompleteRefrigerator:** fungsi ini dipakai untuk mengambil list kulkas beserta detailnya. Aplikasi mengirimkan XML-RPC dengan methodName = rf.getCompleteRefrigerator dan parameter dalam sebuah struct dengan perincian seperti pada Tabel 3.13.

Tabel 3.13 Struktur struct *fungsi rf.getCompleteRefrigerator* request

Key	**Tipe Data**	**Keterangan**

USERNAME	String 20	*Username* yang akan mengambil detail kulkas.
PASSWORD	String 64	*Password* yang telah di-*hash* dengan SHA3-256 untuk autentikasi.

Fungsi ini mengembalikan nilai ke *gateway* dalam sebuah *array of struct* dengan nilai *struct* seperti pada Tabel 3.14.

Tabel 3.14 Struktur struct *(dalam* Array*) fungsi rf.getCompleteRefrigerator* response

Key	**Tipe Data**	**Keterangan**
DEVICEID	String	Device ID kulkas.
JMLTELUR	Int	Jumlah telur yang tersisa di kulkas yang bersangkutan.
SUHU	Int	Suhu kulkas yang terakhir di-*update*. Dalam derajat Celcius.
LASTUPDATE	String	Waktu terakhir kali *update* dalam *long epoch time*.

Berikut adalah contoh *request* yang dikirimkan dari aplikasi.

```
<?xml version="1.0"?>
<methodCall>
<methodName>rf.getCompleteRefrigerator</methodName>
<params>
  <param>
    <value>
      <struct>
        <member>
          <name>USERNAME</name>
          <value><string>tralala</string></value>
        </member>
        <member>
          <name>PASSWORD</name>
          <value><string>086abe9e75baba382d84c1a
            eee6aacec9874aac7799924e286ea000362d
            4db82</string></value>
```

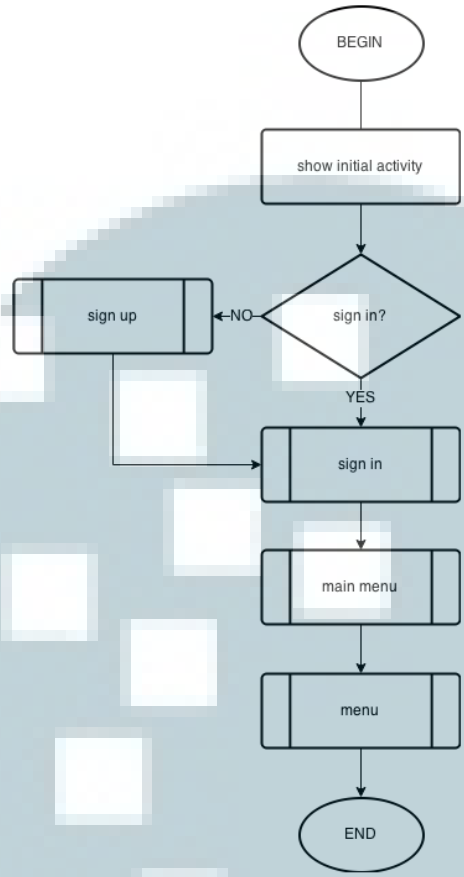
```
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

Berikut adalah contoh *response* yang diberikan server.

```
<?xml version="1.0"?>
<methodResponse>
<params>
  <param>
    <value>
      <array>
        <data>
          <value>
            <struct>
              <member>
                <name>DEVICEID</name>
                <value><string>RF 001</string></value>
              </member>
              <member>
                <name>JMLTELUR</name>
                <value><int>5</int></value>
              </member>
              <member>
                <name>SUHU</name>
                <value><int>4</int></value>
              </member>
              <member>
                <name>LASTUPDATE</name>
                <value><string>1432537435210
                </string></value>
              </member>
            </struct>
          </value>
        </data>
      </array>
    </value>
  </param>
</params>
</methodResponse>eway.
```

3.7 Rancangan Aplikasi Android

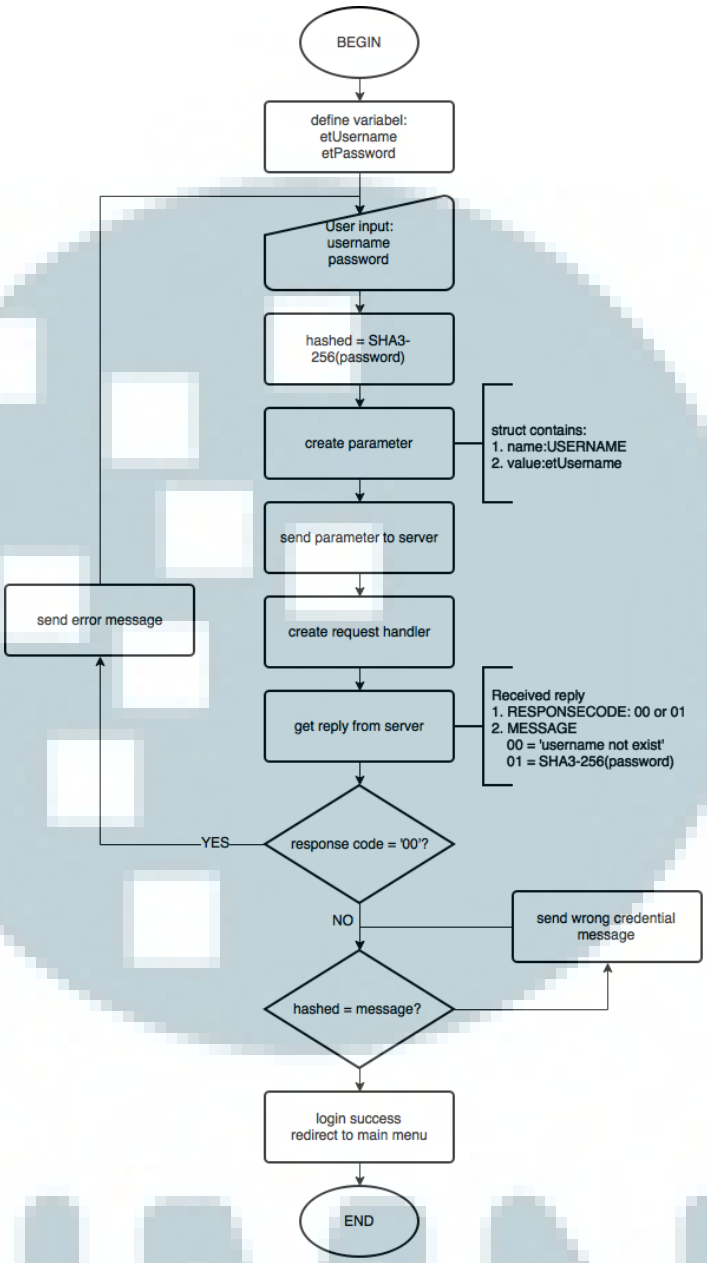
Secara umum, aplikasi Smart Control terdiri dari 6 jenis *activity*, yaitu *sign in* dan *sign up*, *main menu*, *device*, *device control*, dan *setting*. Berikut adalah diagram alir utama aplikasi Smart Control.



Gambar 3.5 Diagram alir keseluruhan aplikasi

1. Halaman *sign in*

   Halaman ini bertugas untuk mengambil input *username* dan *password*. Kedua data ini akan digunakan untuk memanggil suatu fungsi dari *server* dan kemudian mengambil data dari *server* untuk dibandingkan. Apabila *credential* yang dimasukkan sesuai, maka berikutnya pengguna akan dialihkan menuju halaman utama.

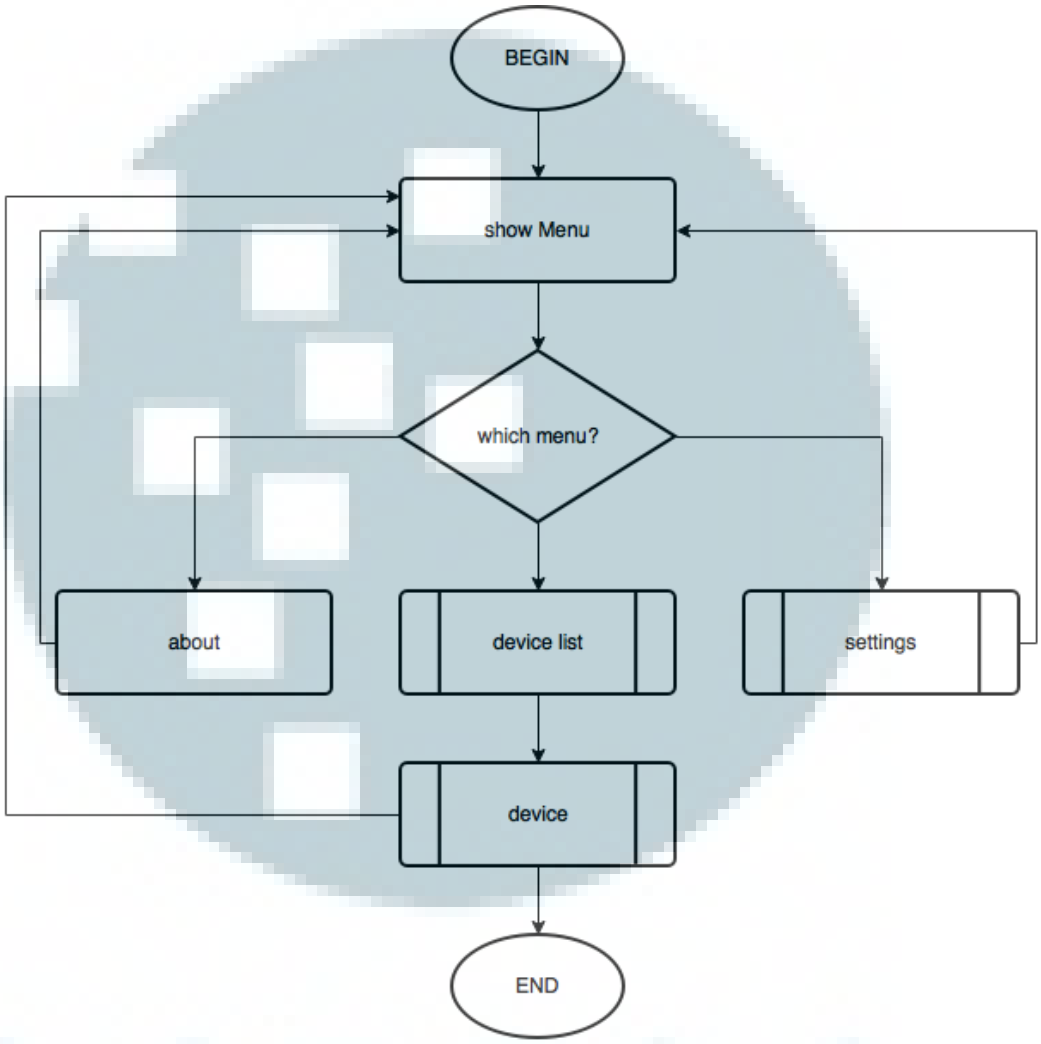


Gambar 3.6 Diagram alir halaman sign in

2. Halaman *main menu*

   Halaman ini berisi tiga pilihan utama, yaitu *device list* yang menampilkan daftar perangkat yang dapat diakses, *setting* untuk mengubah pengaturan pribadi, dan *about* untuk menampilkan informasi singkat mengenai aplikasi. Masing-masing menu *device*

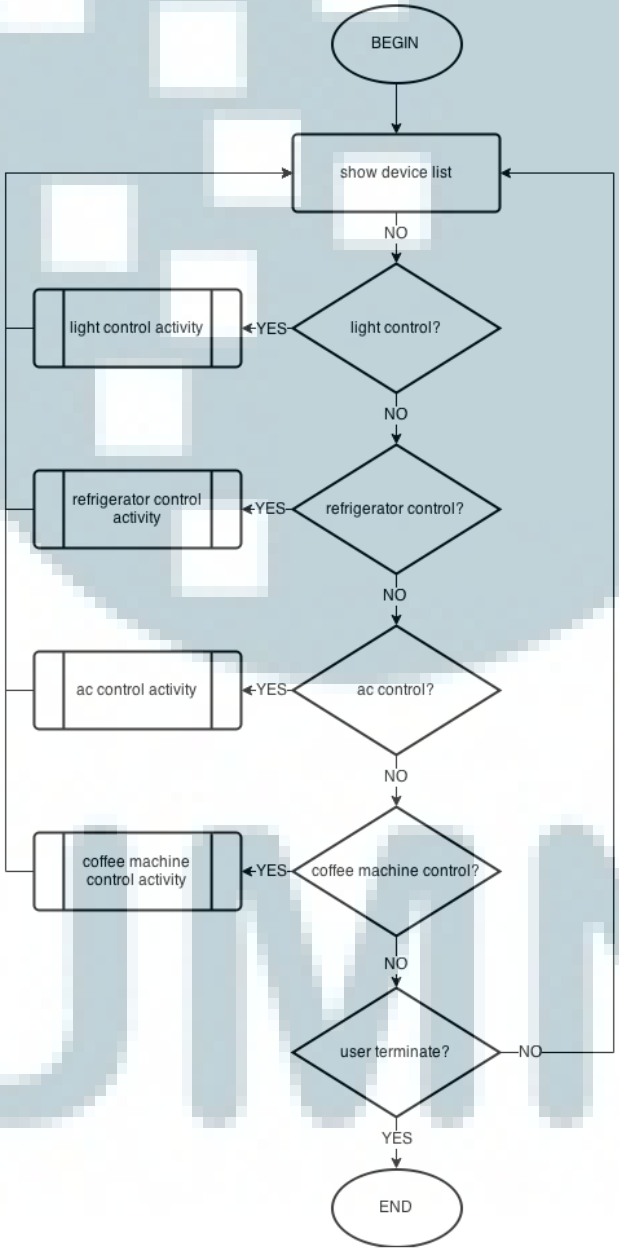
list dan *setting* merupakan *predefined process* yang memiliki diagram alir tersendiri.



Gambar 3.7 Diagram alir halaman utama

3. Halaman *device list*

   Di halaman ini, pengguna dapat memilih perangkat yang ingin diakses. Terdapat 4 pilihan dasar yaitu ac, lampu, mesin pembuat kopi dan kulkas. Masing-masing menu perangkat merupakan *predefined process* tersendiri.



Gambar 3.8 Diagram alir halaman device list

4. Halaman *device*

   Untuk halaman ini, setiap perangkat akan memiliki halaman dan fungsi yang berbeda. Untuk kulkas, hanya akan ditampilkan stok telur, suhu kulkas, dan waktu pembaharuan terakhir. Untuk waktu, format data yang diterima dari *server* adalah dalam bentuk *epoch time*.

3.8 Rancangan Database

Untuk sistem kulkas pintar, digunakan sebuah *database* yang menggunakan mySQL dan diletakkan di *cloud*. Gambar 3.10 merupakan *Entity Relationship Diagram* (ERD) dari *database*, dengan Struktur dari setiap tabel adalah sebagai berikut:

1. Tabel *User*

   Tabel *User* berisi *username* dan *password* yang akan digunakan sebagai pembanding pada saat ada pengguna yang ingin melakukan *sign in* ke aplikasi.

2. Tabel *Device*

   Tabel *Device* memiliki atribut *deviceID* yang unik dan merupakan *primary key* dari tabel ini, *username* yang merupakan *foreign key* yang mengacu ke *username* pada tabel *User*, tipe dan EUI64 *device* yang terdaftar, serta status aktivitasnya.

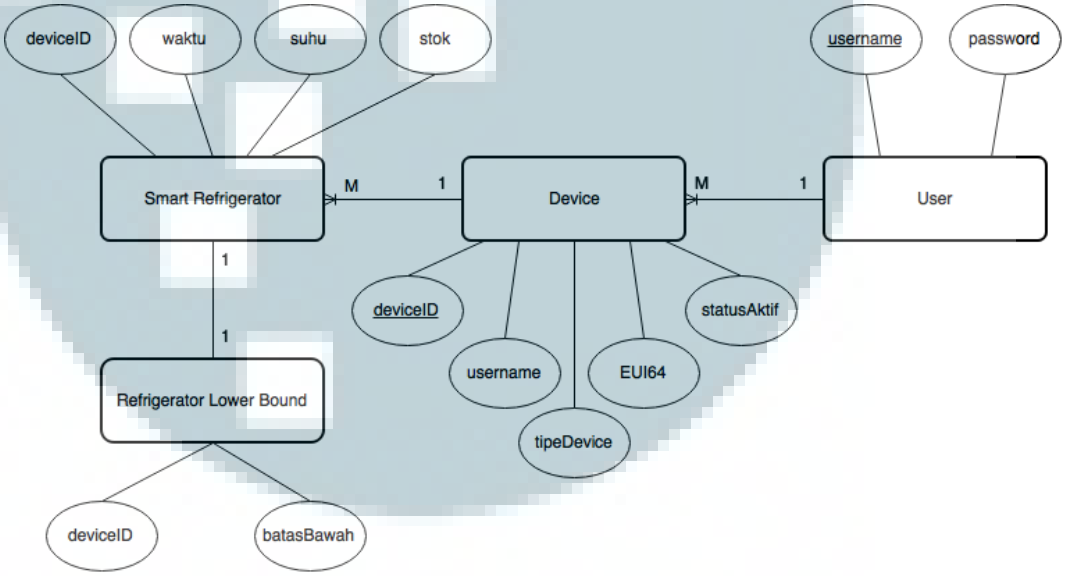
3. Tabel Kulkas

   Tabel ini berisi deviceID (untuk kulkas, deviceID nya adalah RF xxx) yang mengacu pada deviceID pada tabel User, waktu *update* dalam

format *epoch time* dalam satuan detik, stok barang dalam *integer* (dalam asumsi ini batas maksimal 8 buah), dan suhu kulkas dalam *integer*.

4. Tabel KulkasSetting

   Tabel ini berisi deviceID (untuk kulkas, deviceID nya adalah RF xxx) yang mengacu pada deviceID pada tabel User, dan batas bawah jumlah terur dalam *integer*.



Gambar 3.9 Entity Relation Diagram *dari* database k3842690_umn-iot-taskforceDB