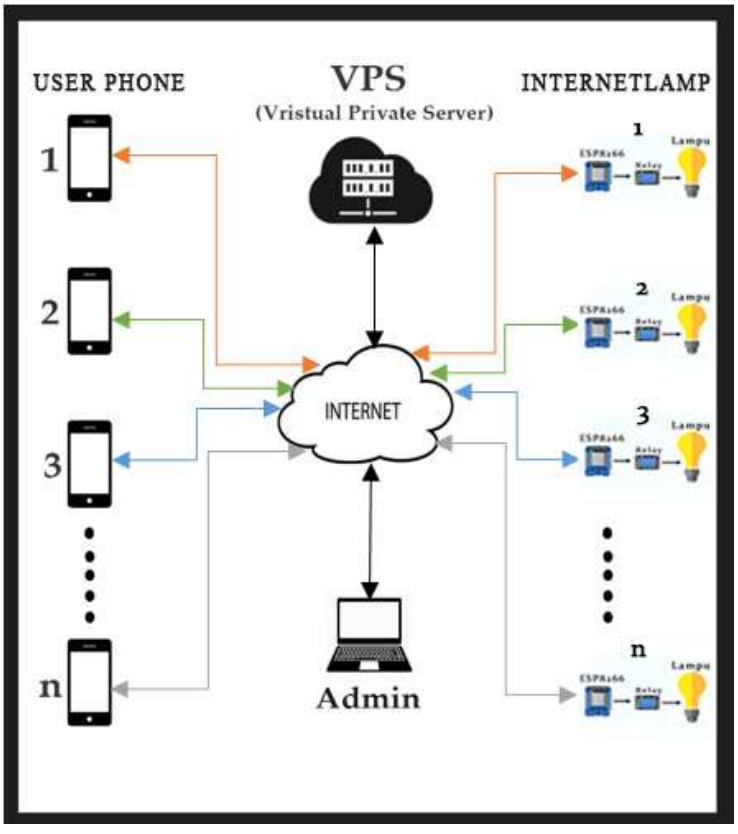
BAB III
PERANCANGAN SISTEM

Dalam tugas akhir ini penulis mencoba membuat layanan *Internet of Things (IoT*) yang bertujuan untuk membantu menghubungkan peralatan yang berada dirumah ke internet agar dapat di kontrol jarak jauh menggunakan *smartphone* dan menghasilkan produk berupa *internet lamp* sebagai alat pengujian sistem. Pada bab ini akan membahas tentang perancangan sistem dan penentuan kebutuhan sistem Layanan IoT menggunakan *Virtual Private Server (VPS)*, serta perancangan aplikasi untuk memberikan *user* alamat API, Token, dan Penyimpanan Data, lalu seluruh informasi tersebut bisa di salin ke email dan juga dapat menambahkan *device internet lamp* dengan mengisikan *serial number* pada aplikasi untuk mengontrol lampu lewat internet. Pada gambar 3-1 berikut ini merupakan gambaran umum sistem secara keseluruhan.



Gambar 3-1 Gambaran Umum Sistem

Keterangan gambar:

1. *User Phone* adalah perangkat *smartphone* yang digunakan menjalankan aplikasi Layanan IoT untuk mengontrol *Internet lamp*
2. *Admin* dapat mengakses langsung ke basis data menggunakan leptop atau komputer untuk mengatur *user* yang sudah terdaftar.
3. Internet merupakan tempat terhubungnya berbagai mesin computer yang mengolah informasi, baik berupa server, komputer pribadi dan lain sebagainya[11]. Internet menghubungkan antara *User Phone*, VPS, dan *Internet lamp*.
4. *Virtual Private Server (VPS)* merupakan perangkat yang menjadi sistem dan wadah untuk menyimpan data serta penghubung antara *User Phone* degan *Internet lamp* melalui Internet. Data-data tersebut terdiri dari data *Device Internet lamp* dan data *User*.
5. *Internet lamp* merupakan lampu yang dapat dikontrol melalui jaringan internet yang telah dipasang API dan memiliki *serial number*.

Terdapat beberapa komponen yang digunakan dalam perancangan dan pembuatan implementasi Layanan IoT menggunakan VPS, komponen tersebut terbagi atas perangkat keras dan perangkat lunak.

3.1 Perangkat Keras

Pada bagian ini menjelaskan perangkat keras yang dibutuhkan untuk membangun sistem Layanan IoT.

3.1.1 Kebutuhan Perangkat Keras

Perangkat keras yang dibutuhkan untuk menjalankan aplikasi yang akan dibangun dapat dilihat pada Tabel 3-1.

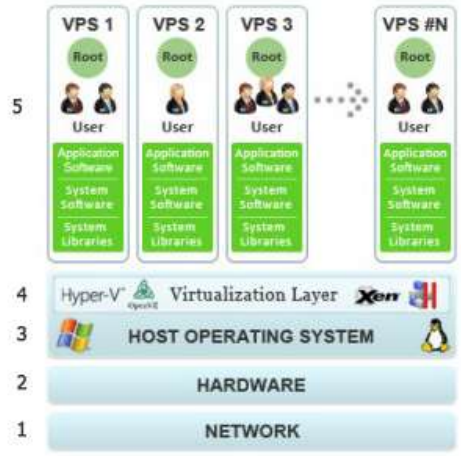
Tabel 3-1 Kebutuhan Perangkat Keras

No	Perangkat Keras	Jumlah	Spesifikasi
1	Phone *(Smartphone)*	3	➢ Prosesor Phone Quad Core 1.2 GHz. ➢ Ukuran Layar 5.0 Inch

			➢ Memori RAM 2 GB. ➢ Memori Internal 8 GB
2	VPS	1	➢ Prosesor VPS 1 CPU 2.50 GHz Memori RAM 2 GB ➢ Hardisk 30 GB ➢ Bandwidth Unmetered
3.	*Internet lamp*	3	➢ Wemos ➢ Relay ➢ Lampu

3.1.2 Perancangan Virtual Private Server (VPS)

Perancangan VPS dalam sistem ini berfungsi sebagai *gateway* penghubung antara alat listrik yang berada di rumah dengan dunia luar agar dapat di kontrol menggunakan *smartphone*. Gambar 3-2 adalah gambaran umum perancangan VPS:



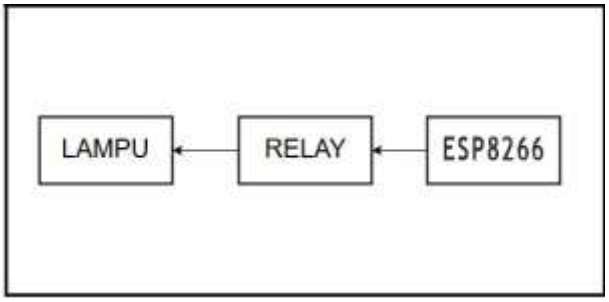
Gambar 3-2 Gambaran Umum Perancangan VPS

Keterangan:

1. Netwrok berfungsi unutk menghubungkan server ke internet.
2. Spesifikasi Hardware yang di gunakan pada server.
3. Sistem Oprasi yang dugunakan pada server.
4. Aplikasi yang digunakan unutk menginstal Sistem Oprasi untuk VPS pada server.
5. Beberapa VPS yang telah terinstal.

3.1.3 Perancangan *Device Internet lamp*

Internet lamp sebagai implementasi sistem Layanan IoT, dibangun menggunakan Microcontroller ESP8266 sebagai alat untuk menghubungkan lampu ke internet, agar lampu dapat di kontrol melalui internet dan relay sebagai saklar untuk menghidupkan atau mematikan lampu.

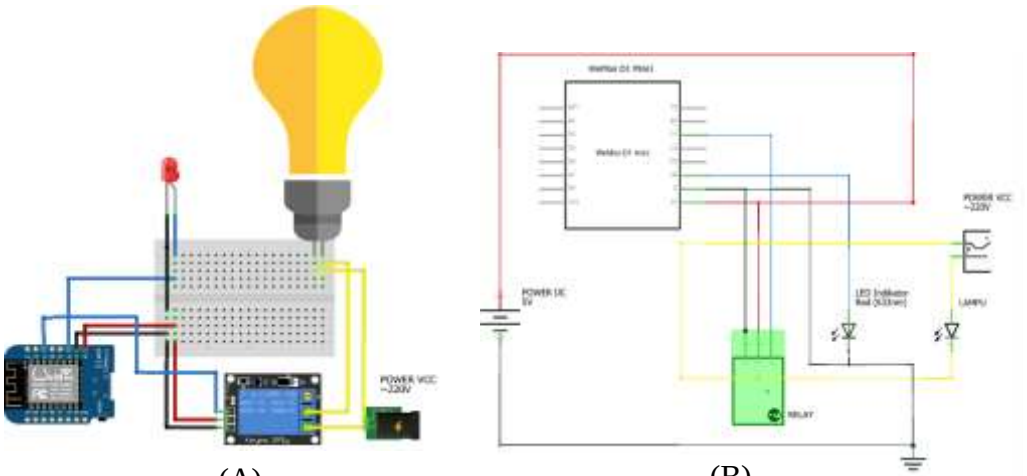


Gambar 3-3 Blok Diagram *Internet lamp*

Keterangan:

1. Lampu yang digunakan adalah lampu yang beredar di pasaran.
2. Relay berfungsi sebagai saklar untuk menghidupkan atau mematikan lampu yang di kontrol oleh ESP8266.
3. ESP8266 yang berfungsi menghubungkan lampu dengan internet.

Pada gambar 3-4 adalah schematic perancangan yang dibuat untuk *Device Internet lamp.*



(A) (B)

Gambar 3-4 (A) dan (B) *schematic* perancangan *device Internet lamp.*

3.2 Perangkat Lunak

Pada bagian ini menjelaskan perangkat lunak yang dibutuhkan untuk membangun sistem Layanan IoT.

3.2.1 Perangkat Lunak Pendukung

Perangkat lunak Layanan IoT yang dibangun membutuhkan perangkat lunak pendukung sehingga perangkat lunak Layanan IoT dapat berjalan dengan baik. Adapun perangkat lunak pendukung yang dibutuhkan untuk menjalankan perangkat lunak Layanan IoT yang terdapat pada tabel 3-2 adalah:

Tabel 3-2 Kebutuhan Perangkat Lunak

No	Perangkat Lunak	Keterangan
1	Sistem Oprasi *Phone*	Android versi 5.0 Lollipop
2	Sistem Oprasi *VPS*	CentOS 7
3	Editor Aplikasi	a. Android Studio 2.2.2 b. Sublime Text c. Arduino IDE
4	Basis Data	MongoDB
5	Admin	Robomongo
6	Aplikasi Back-End Server	Node Js
7	*Process Manager Untuk Node Js*	PM2
8	*Remot VPS via SSH*	PuTTy
9	*Download/upload* file ke *VPS*	WinSCP
10	Pengujian REST API dan Penambahan data *Internet lamp* ke Server	Postman

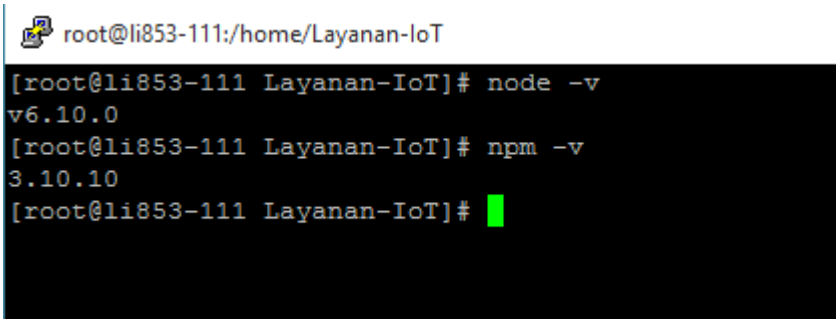
3.2.2 Pembangunana Perangkat lunak *Back-End* Pada VPS

Pembangunan perangkat lunak *back-end* pada VPS dilakukan menggunakan Node Js untuk konfigurasi menggunakan *software* Sublime Text. Dalam pembangunan aplikasi back-end terdapat beberapa fungsi yaitu login, register, RESTful API untuk membuat alamat *API user*, JSON web token untuk membuat token, *authentication*, inisialisasi basis data, menyimpan data ke basis data, mengambil data dari basis data, mengubah data di basis data dan menampilkan profil yang berisi informasi alamat *API user*, *token*, dan basis data yang di berikan pada *user* untuk menyimpan data.

Adapun langkah-langkah yang harus di kerjakan sebgai berikut:

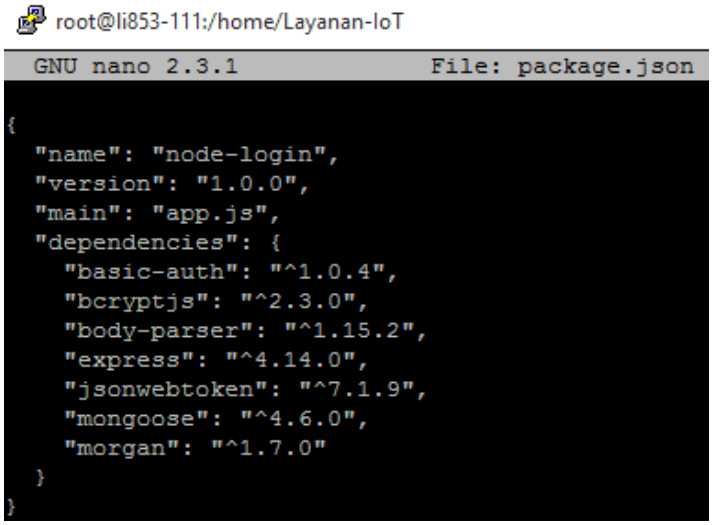
1. *Install* terlebih dahulu Node Js pada VPS dengan cara membuat folder untuk tempat install Node Js tersebut lalu eksekusi perintah berikut:
   - `curl --silent --location https://rpm.nodesource.com/setup_6.x | bash -`
   - `yum -y install nodejs`
2. Setelah *terinstall* masuk ke dalam folder Node Js tersebut lalu *install* NPM berisi *package* atau *module* untuk aplikasi NodeJS dengan perintah:
   - `NPM Install`
3. Cek untuk memastika *installasi* berjalan dengan baik dengan perintah:
   - `node -v`
   - `npm -v`

   Jika berjalan dengan baik maka akan terlihat versi dari aplikasi tersebut.



Gambar 3-5 Instalasi Node Js dan NPM

4. Kemudaian membuat file baru bernama package.json untuk mendefinisikan paket apa saja yang diperluka oleh aplikasi Node Js.



Gambar 3-6 Mendefinisikan paket yang di perlukan

Keterangan paket yang diperlukan oleh Node Js:

a) Basic-auth.

Basic-auth berfungsi untuk *authentication* pada saat *login*. *Email* dan *Password* yang dikirim oleh *user* berupa *string* Base64 *encoded* dikirim melalui *header*, lalu *didecode* oleh *Basic-auth*.

b) Bcrytpjs

Bcrytpjs digunakan untuk membuat dan memverifikasi *hash password* menggunakan algoritma *hash bcrypt*.

c) Body-parser

Body-parser digunakan untuk *parsing JSON body* di setiap permintaan.

d) Express

Express adalah *framework* yang digunakan untuk membuat RESTful API.

e) JSON Web Token

Jsonwebtoken berfungsi membuat *JSON Web Token* yang digunakan untuk akses API.

f) Mongoose

Mongoose digunakan untuk koneksi ke *database* MongoDB.

g) Morgan

Morgan digunakan untuk mencetak *log* di terminal untuk setiap permintaan HTTP.

5. Membuat Fungsi *Register*.

Register adalah syarat untuk *user* mendapatkan layanan dari sistem ini. Jika *user* telah *registrasi* ke sistem maka *user* deberikan akses ke sistem. Berikut potongan *source code* program Register pada *VPS.*

```
'use strict';
const user = require('../models/user');
const bcrypt = require('bcryptjs');
exports.registerUser = (name, email, password, stringdata, stringdata2, stringdata3,
stringdata4, serialnumber, serialnumber2, serialnumber3, angkaint) =>
  new Promise((resolve,reject) => {
      const salt = bcrypt.genSaltSync(10);
          const hash = bcrypt.hashSync(password, salt);
          const newUser = new user({
                  name: name,
                  email: email,
                  stringdata: stringdata,
                  stringdata2: stringdata2,
                  stringdata3: stringdata3,
                  stringdata4: stringdata4,
                  serialnumber: serialnumber,
                  serialnumber2: serialnumber2,
                  serialnumber3: serialnumber3,
                  angkaint: angkaint,
                  hashed_password: hash,
                  created_at: new Date()});
newUser.save()
.then(() => resolve({ status: 201, message: 'User Registered Sucessfully !' }))
.catch(err => {if (err.code == 11000)
{reject({ status: 409, message: 'User Already Registered !' });} else
{reject({ status: 500, message: 'Internal Server Error !' });}});});
```

6. Membuat Fungsi Login.

Apabila *user login* maka sistem akan memberikan alamat *API, token, database.* Kompenen-kompenen tersebut dijadikan bahan membangun *software* oleh *user* untuk untuk keperluannya. *User* juga dapat menambahkan *internet lamp* dengan mengisi *serial number* apabila *user* memiliki modul *internet lamp* yang telah disediakan. Berikut potongan *source code* program *Login* pada *VPS.*

```
'use strict';
const user = require('../models/user');
const bcrypt = require('bcryptjs');
exports.loginUser = (email, password) =>
new Promise((resolve,reject) => {
  user.find({email: email})
  .then(users => {
          if (users.length == 0) {
                  reject({ status: 404, message: 'User Not Found !' });
          } else {
                  return users[0];
          }
  })
  .then(user => {
          const hashed_password = user.hashed_password;
          if (bcrypt.compareSync(password, hashed_password)) {
          resolve({ status: 200, message: email });
          } else {
          reject({ status: 401, message: 'Invalid Credentials !' });
          }
  })
  .catch(err => reject({ status: 500, message: 'Internal Server Error !' }));});
```

7. Membuat fungsi RESTful API.

RESTful API berfungsi sebagai jembatan antara *user phone* dengan *internet lamp* melalui jaringan internet. Email yang didaftarkan user dipakai menjadi parameter ID. Berikut potongan *source code* program RESTful API pada *VPS.*

```
router.post('/users', (req, res) => {
  const name = req.body.name;
  const email = req.body.email;
  const password = req.body.password;
  const stringdata = req.body.stringdata;
        if (!name || !email || !password || !stringdata || !name.trim() ||
        !email.trim() || !password.trim() || !stringdata.trim()) {
```

```
        res.status(400).json({message: 'Invalid Request !'});
  } else {
        register.registerUser(name, email, password, stringdata)
        .then(result => {
            res.setHeader('Location', '/users/'+email);
            //const token = jwt.sign(result, config.secret, { expiresIn: 1440 });
            res.status(result.status).json({ message: result.message})
        })
        .catch(err => res.status(err.status).json({ message: err.message }));
  }
});
```

8. Membuat Fungsi JSON Web Token.

   JSON Web Token menghasilkan token yang berfungsi sebagai *validasi* akses dari alamat API *user.* Bertujuan sebagai keamanan dalam jaringan, apabila *user* telah memiliki API maka untuk membuatnya berfungsi sebagai penghubung antara *user phone* dengan *internet lamp* dibutuhkan token. Berikut potongan *source code* program JSON Web Token pada *VPS.*

```
const jwt = require('jsonwebtoken');
function checkToken(req) {
        const token = req.headers['x-access-token'];
        if (token) {
                try {
                                var decoded = jwt.verify(token, config.secret);
                                return decoded.message === req.params.id;
                } catch(err) {
                        return false;
                }
        } else {return false;}}
```

9. Membuat Fungsi *Authentication.*

   Sebelum *user* mendapatkan token terlebih dahulu user harus melewati tahap *authentication. Authentication* yaitu tahap validasi akun user yang telah didaftarkan ke sistem, parameternya ada dua yaitu *email* dan *password.* Apabila user lolos *authentication* akan mendapatkan token, sebaliknya jika tidak lolos maka *user* tidak akan mendapatkan token. Berikut potongan *source code* program JSON Web Token pada *VPS.*

```
router.post('/authenticate', (req, res) => {
  const credentials = auth(req);
  if (!credentials) {
        res.status(400).json({ message: 'Invalid Request !' });
  } else {
```

```
        login.loginUser(credentials.name, credentials.pass)
        .then(result => {
        const token = jwt.sign(result, config.secret, { expiresIn: 1440 });
 res.status(result.status).json({ message: result.message, token: token });
})
.catch(err => res.status(err.status).json({ message: err.message }));}});
```

10. Inisialisasi Basis Data.

Pada sistem ini dibutuhkan tempat penyimpanan data di *VPS* untuk menyimpan informasi akun user, data user dan data modul *internet lamp*. Basis data menggunakan MongoDB. Berikut potongan *source code inisialisasi* basis data pada *user* dan *internet lamp VPS.*

a) *Source code inisialisasi* basis data untuk *user*

```
'use strict';
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const userSchema = mongoose.Schema({
           name                          : String,
           email                         : String,
           stringdata                    : String,
           stringdata2                   : String,
           stringdata3                   : String,
           stringdata4                   : String,
           serialnumber                  : String,
           serialnumber2                 : String,
           serialnumber3                 : String,
           devdata                       : String,
           checkdevice                   : String,
           angkaint                      : Number,
           hashed_password               : String,
           created_at                    : String,
           temp_password                 : String,
           temp_password_time            : String});
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://perbangga:1993abrizen@139.162.11.111:27093/user_d
ata');
module.exports = mongoose.model('user', userSchema);
```

b) *Source code inisialisasi* basis data untuk *internet lamp.*

```
'use strict';
const user = require('../models/user');
const bcrypt = require('bcryptjs');
exports.registerDevice = (email, devdata, checkdevice) =>
         new Promise((resolve,reject) => {
                   const newUser = new user({
                             email: email,
                             devdata: devdata,
                             checkdevice: checkdevice,
                             created_at: new Date()
                   });
                   newUser.save()
.then(() => resolve({ status: 201, message: 'User Registered Sucessfully !'
}))
         .catch(err => {
         if (err.code == 11000) {
         reject({ status: 409, message: 'User Already Registered !' });
         } else {
         reject({ status: 500, message: 'Internal Server Error !' });}});
         });
```

11. Menyimpan Data ke Basis Data.

Dalam Bahasa pemrograman ada istilah *CRUD* yaitu *create, read, update, and delete.* Dalam sistem ini *CRUD* diimplementasikan pada Node Js menggunakan bahas Javascript. Untuk penyimpanan ke basis data menggunakan fungsi *create* atau dalam Javascript digunakan istilah *POST.* Berikut potongan *source code* program menyimpan ke basis data pada *VPS.*

```
'use strict';
const user = require('../models/user');
const bcrypt = require('bcryptjs');
exports.registerUser = (name, email, password, stringdata, stringdata2,
stringdata3, stringdata4, serialnumber, serialnumber2, serialnumber3, angkaint) =>
    new Promise((resolve,reject) => {
        const salt = bcrypt.genSaltSync(10);
               const hash = bcrypt.hashSync(password, salt);
               const newUser = new user({
                          name: name,
                          email: email,
                          stringdata: stringdata,
                          stringdata2: stringdata2,
                          stringdata3: stringdata3,
                          stringdata4: stringdata4,
                          serialnumber: serialnumber,
                          serialnumber2: serialnumber2,
                          serialnumber3: serialnumber3,
                          angkaint: angkaint,
                          hashed_password: hash,
                          created_at: new Date()
               });
               newUser.save()
    .then(() => resolve({ status: 201, message: 'User Registered Sucessfully !' }))

    .catch(err => {

    if (err.code == 11000) {

               reject({ status: 409, message: 'User Already Registered !' });

    } else {

               reject({ status: 500, message: 'Internal Server Error !' });}});});
```

12. Membaca Data dari Basis Data.

Membaca data dari basis data menggunakan fungsi *read* atau dalam istilah Javascript disebut *GET*. Dalam sisitem ini *GET* berfungsi untuk menampilkan profil *user* yang berisi informasi alamat *API user*, *token* dan membaca data yang dikirim *internet lamp* ke VPS ataupun sebaliknya *internet lamp* membaca data yang dikirim ke VPS. Berikut potongan *source code* program membaca data dari basis data pada *VPS.*

```
'use strict';
const user = require('../models/user');
exports.getProfile = email =>
    new Promise((resolve,reject) => {
               user.find({ email: email }, { name: 1, serialnumber:1,
serialnumber2:1, serialnumber3:1, stringdata:1, stringdata2:1, stringdata3:1,
```

```
stringdata4:1, devdata:1, checkdevice:1, angkaint:1, email: 1, created_at: 1, _id:
1 })
                .then(users => resolve(users[0]))
                .catch(err => reject({ status: 500, message: 'Internal Server Error
!' }))

    });
```

13. Memperbarui Data di Basis Data.

    Memperbarui data di basis data menggunakan fungsi *Update* atau dalam istilah Javascript disebut *PUT*. Dalam sisitem ini *PUT* berfungsi untuk memperbarui data di basis data yang telah tersimpan saat *user register*. Berikut potongan *source code* program memperbarui data dari basis data pada *VPS*.

```
'use strict';
const user = require('../models/user');
const bcrypt = require('bcryptjs');
exports.putDevdata = (email, devdata) =>
    new Promise((resolve,reject) => {
                user.find({email: email})
                .then(users => {
                            let user = users[0];
                            user.devdata = devdata;
                            user.save();
                            if (users.length == 0) {
    reject({ status: 404, message: 'Data Not Found !' });
                            } else {
          resolve({ status: 200, message: devdata });
                            }
                })
    .then(user => resolve({ status: 200, message: 'Updated Sucessfully !' }))
    .catch(err => reject({ status: 500, message: 'Internal Server Error !' }));

    });
```

Setelah semua konfigurasi pada Node Js selesai, maka akan terbentuk struktur projek yang terlihat pada gambar 3-7 sebagai berikut:



Gambar 3-7 Struktur Projek Perangkat lunak Layanan IoT Pada VPS

3.2.3 Instalasi Basis Data MongoDB Pada VPS

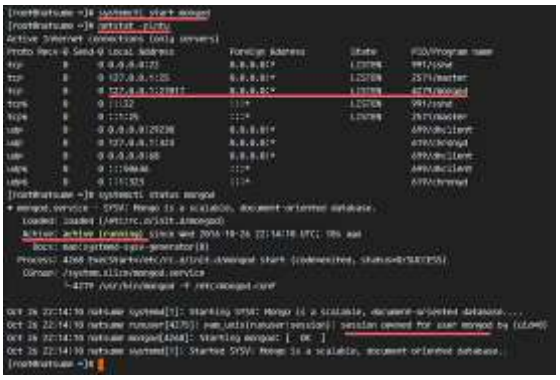
MongoDB berfungsi untuk menyimpan data *user* dan data *internet lamp.* MongoDB menggunakan struktur data JSON untuk menyimpan datanya. Data pada MongoDB dikenal sebagai *document*. MongoDB menggunakan BSON sebagai format penyimpanan data untuk *document*.

Berikut langkah-langkah untuk instalisai MongoDB pada VPS:

1. Pertama harus masuk ke folder *'yum.repos.d'* untuk menambahkan *repository* pada Centos dan buat *repository* baru.
   - `cd /etc/yum.repos.d/`
   - `nano mongodb-org-3.2.repo`
2. Lalu salin kode berikut:

```
[mongodb-org-3.2]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-
org/3.2/x86_64/gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.2.asc
```

3. Setelah itu *Save* file dan keluar text editor.
4. Berikutnya, setelah menambahkan *repository* lalu *install* mongoDB menggunakan perintah *yum,* mengaktifkan mongoDB menggunakan perintah *systemcl start,* cek port mongoDB menggunakan *netstat*, dan cek mongoDB aktif menggunakan perintah *systemctl status.*
   - `yum -y install mongodb-org`
   - `systemctl start mongod`
   - `netstat -plntu`
   - `systemctl status mongod`



Gambar 3-8 Instalasi MongoDB dan Status MongoDB aktif

5. Membuat keamanan *Administrator User* pada basis data MongoDB.

   Buka mongo *Shell* dengan perintah:

   - `mongo`

   Pilih basis data yang akan dipakai:

   - `use user_data`

   Buat *user* akses pada basis data yang akan digunakan dengan perintah berikut:

```
db.createUser(
  {
   user: "perbangga",
   pwd: "inipasaword",
   roles: [ "readWrite"]
  })
```

   Jadi hanya user perbangga yang dapat mengakses basis data tersebut.

6. Mengaktifkan *Authentication* di MongoDB.

   Masuk ke folder *'/etc/init.d/'* dan edit *'mongod'* file:

   - `cd /etc/init.d/`
   - `vim mongod`

   Tambahkan *'–auth'* pada basri *'OPTION'*:

   - `OPTIONS=" --auth -f $CONFIGFILE"`

   Lalu *Save file*

   Reload systmcl dan restart mongoDB:

   - `systemctl daemon-reload`
   - `systemctl restart mongod`

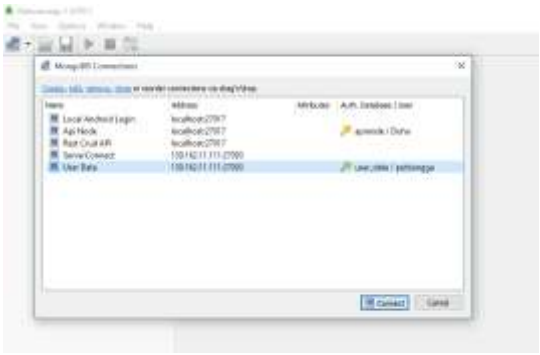
Instalisasi mongoDB telah selesai, basis data telah siap dipakai dengan nama *user_data*. Untuk mengakses basis data tersebut hanya *admin* yang bisa karena harus memiliki *user* dan *password* yang telah konfigurasi tidak sembarang orang yang dapat mengaksesnya.

3.2.4 Mengakses Basis Data MongoDB menggunakan Robomongo

Robomongo adalah management tools untuk mongoDB memudahkan proses management basis data di MongoDB dan dapat digunakan dibeberapa Sistem Oprasi seperti Windows, Linux dan MAC OS. Robomongo digunkan oleh *admin* untuk mengatur data *user* dan data *internet lamp*. Memakai Robomongo *admin* dapat menghapus, mengedit, *update* dan *input* data ke basis data di MongoDB.

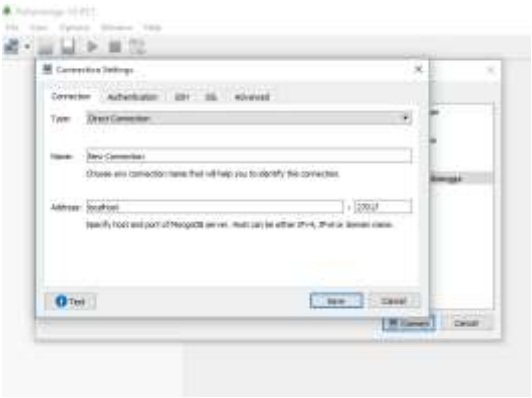
Berikut cara mengakses mongoDB mengggunakan Robomongo:

1. Pertama instal Robomongo pada komputer atau leptop admin.



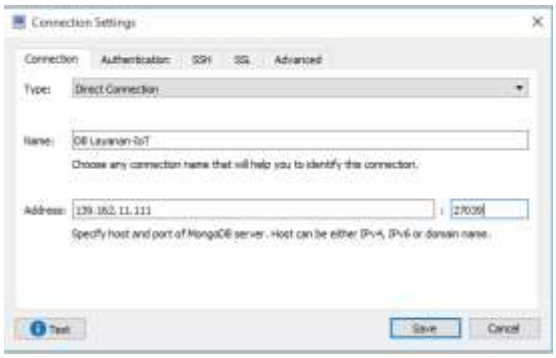
Gambar 3-9 Tampilan Robomongo setelah di instal

2. Untuk mengakses basis data yang telah dibuat di mongoDB harus membuat koneksi baru di menu *create*. Lalu akan muncul halaman baru dengan menu *Connection, Authentication, SSH, SSL, Advance*.



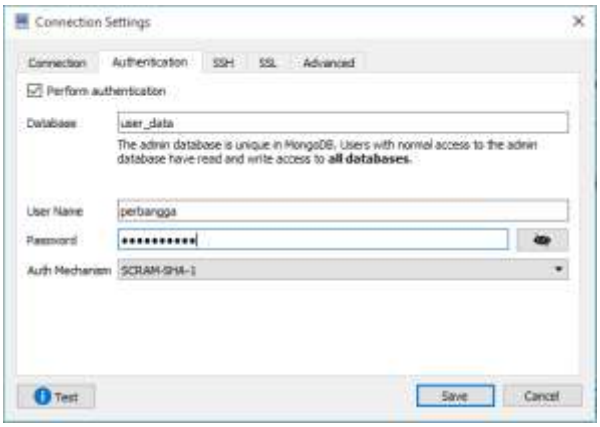
Gambar 3-10 Halaman menu *create* Robomongo

3. Isi menu Connection dengan alamat IP Server dan port dimana MongoDB berjalan.



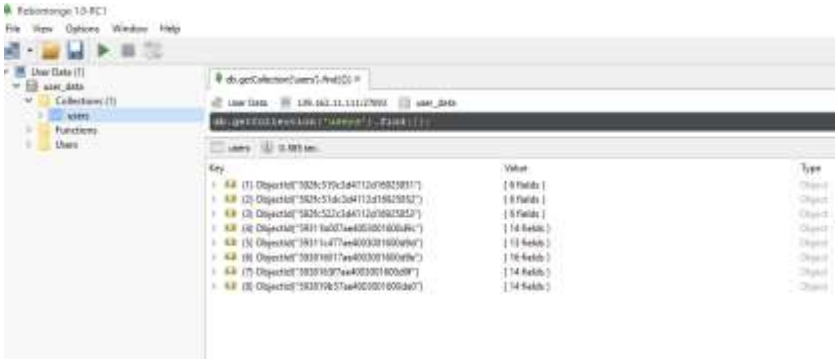
Gambar 3-11 Mengisi IP Server dan Port pada Robomongo

4. Setelah mengisi IP Server lanjut ke menu *authentication* untuk menerima hak akses dari server dengan mengisikan nama basis data, *user* dan *password* yang telah dikonfigurasi.



Gambar 3-12 Meminta hak akses ke Server

5. Jika berhasil maka akan tampil basis data dan isinya



Gambar 3-13 Berhasil Menerima Hak Akses dari Server

3.2.5 Kebutuhan Pengguna

Perangkat lunak Layanan IoT menggunakan Android sebagai *interface* untuk mengakses perangkat lunak *back-end* yang telah dikonfigurasi pada VPS. Pengguna yang mengoperasikan Layanan IoT berbasis Android adalah *User*. Tabel 3-3 menunjukan analisis kebutuhan pengguna.

Tabel 3-3 Kebutuhan Pengguna Aplikasi

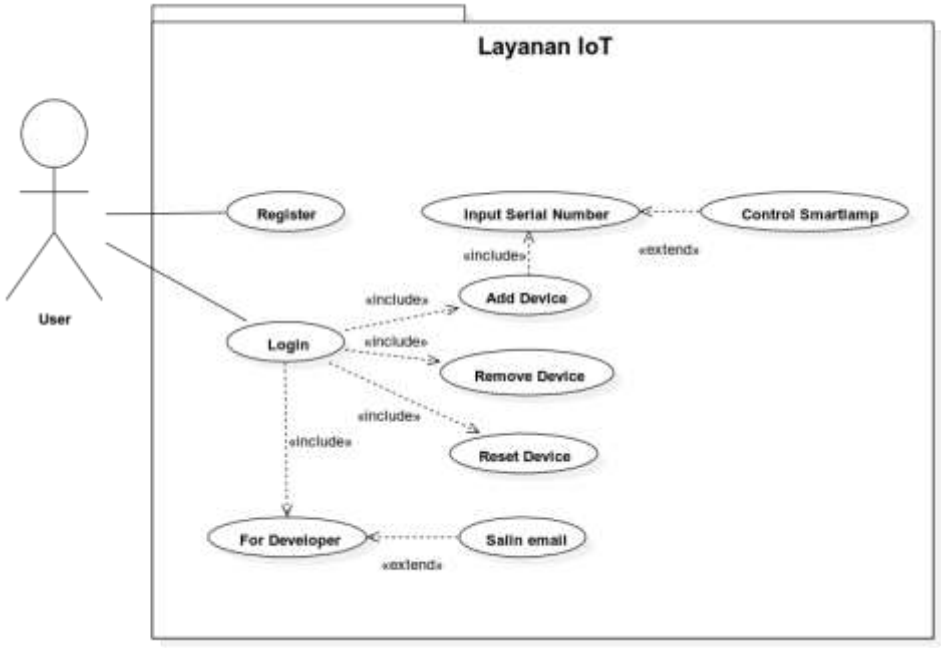
No	pengguna	Keterangan	Kemampuan
1	User	Dapat register dan login. Setelah Login user dapat melihat alamat API, token, *database*, serta menyalin semua informasi ke email dan dapat menambahkan modul *Internet lamp* dengan mengisi serial number.	Mempu menggunakan smartphone, memahami penggunaan aplikasi berbasis android

3.2.6 Perancangan Perangkat Lunak Android

Dalam merancang perangkat lunak ini menggunakan UML (*Unified Modelling Language*)[12]. Pada dasarnya aplikasi mempunyai empat fungsi yaitu *Register, Login*, mampilkan informasi API, dan mengontrol *Internet lamp.*

3.2.6.1 Use Case Diagram

Use case diagram merupakan suatu diagram yang menggambarkan fungsionalitas yang dimiliki oleh suatu sistem beserta aktor-aktor yang terlibat di dalamnya[13]. Gambar 3-14 merupakan *use case* diagram aplikasi Layanan IoT.



Gambar 3-14 Use Case Diagram Aplikasi Layanan IoT

1. Definisi Aktor

Definisi aktor mendeskripsikan peranan aktor yang ada pada sistem. Definisi aktor pada sistem Layanan IoT dapat dilihat pada Tabel 3.4.

Tabel 3-4 Definisi Aktor

No	Aktor	Deskripsi
1	User	Orang yang memiliki hak akses untuk mendapatkan informasi API dan dapat melakukan pengontrolan *Internet lamp* apabila memiliki *serial number*.

2. Definisi *Use Case*

Definisi *use case* mendeskripsikan setiap *use case* yang terdapat pada *use case* diagram. Pada Tabel 3-5 merupakan definisi *use case* pada sistem yang dibuat.

Tabel 3-5 Definisi *Use Case*

No	*Use case*	Deskripsi
1	*Register*	Merupakan proses *user* mendaftar pada aplikasi Layanan IoT.
2	*Login*	Merupakan proses *user* masuk ke aplikasi Layanan IoT
3	*For Developer*	Merupakan proses *user* mendapatkan informasi API.
4	Salin *email*	Merupakan proses untuk menyalin informasi API ke *email*
5	*Add Device*	Merupakan proses untuk menambah *device internet lamp*
6	*Remove Device*	Merupakan proses untuk menghapus salah satu *device internet lamp* yang sudah di daftarkan
7	*Reset Device*	Merupakan proses untuk menghapus semua *device internet lamp* yang sudah di daftarkan

8	*Input Serial Number*	Merupakan prose *menginput serial number* untuk dapat mengontrol *internet lamp*
9	*Control Internet lamp*	Merupakan proses untuk mengontrol *internet lamp*

3. Use Case Secenario

Use case scenario merupakan deskripsi urutan langkah-langkah dalam proses sistem Layanan IoT, baik yang dilakukan oleh aktor terhadap sistem maupun yang dilakukan oleh sistem terhadap aktor.

Berikut *Use Case Secenario* dari Aplikasi Layanan IoT:

1) Nama *Use Case* : Register

Deskripsi: Menjelaskan proses *user registrasi* pada *form register.*

Aktor: *User*

Tabel 3-6 *Use Case Secenario Register*

Aktor	Reaksi Sistem
Sekenario Normal	
1. Masukan *username*	
2. Masukan *email*	
3. Masukan *Password*	
	4. Menyimpan data ke basis data.

2) Nama *Use Case*: Login

Deskripsi: Menjalankan prose *user login* pada *form login.*

Aktor: *User*

Tabel 3-7 *Use Case Secenario Login*

Aktor	Reaksi Sistem
Sekenario Normal	
1. Masukan *username* dan *password*	

	2. Validasi *username* dan *password.* 3. Masuk ke halaman utama
Sekenario Alternatif	
1. Masukan *username* dan *password*	
	2. Validasi *username* dan *password.*
	3. Menampilkan pesan *username* atau *password invalid*, *login* gagal.
4. Memasukan *username* dan *password* yang *valid*	
	5. Validasi *username* dan *password.*
	6. Masuk ke halaman utama

3) Nama *Use Case*: For Developer

Deskripsi: Menjalankan proses untuk mendapatkan informasi API.

Aktor: *User*

Tabel 3-8 *Use Case Secenario For Developer*

Aktor	Reaksi Sistem
Sekenario Normal	
1. Aktor login pada aplikasi	
	2. Menampilkan halaman utama.
3. Memilih *For Developer*	
	4. Menampilkan alamat API, *string* basis data, dan token

4) Nama *Use Case:* Salin Email

Deskripsi: Menjlankan prose untuk menyalin informasi API ke *email.*

Aktor : *User*

Tabel 3-9 *Use Case Secenario* Email

Aktor	Reaksi Sistem
Sekenario Normal	
1. Aktor login pada aplikasi	
	2. Menampilkan halaman utama.
3. Memilih *For Developer*	
	4. Menampilkan alamat API, *string* basis data, dan token.
5. Memilih Salin *email.*	
	6. Menyalin informasi API pada *email.*

5) Nama *Use Case*: Add Device

Deskripsi: Menjalankan proses untuk menambah *internet lamp.*

Aktor: *User*

Tabel 3-10 *Use Case Secenario Add Device*

Aktor	Reaksi Sistem
Sekenario Normal	
1. Aktor login pada aplikasi	
	2. Menampilkan halaman utama.
3. Memilih *Add Device*	
	4. Menambah device *internet lamp*

6) Nama *Use Case*: Remove Device

Deskripsi: Menjalankan proses menghapus salah satu *device internet lamp* yang sudah di daftarkan.

Aktor: *User*

Tabel 3-11 *Use Case Secenario Remove Device*

Aktor	Reaksi Sistem
Sekenario Normal	
1. Aktor login pada aplikasi	
	2. Menampilkan halaman utama.
3. Memilih *Remove Device*	
	4. Menghapus salah satu device *internet lamp*

7) Nama *Use Case*: Reset Device

Deskripsi: Menjalankan proses unutk menghapus semua *device internet lamp* yang sudah di daftarkan.

Aktor: *User*

Tabel 3-12 *Use Case Secenario Reset Device*

Aktor	Reaksi Sistem
Sekenario Normal	
1. Aktor login pada aplikasi	
	2. Menampilkan halaman utama.
3. Memilih *Reset Device*	
	4. Menghapus semua device *internet lamp*

8) Nama *Use Case*: Input Serial Number

Deskripsi: Menjalankan proses untuk menginput serial number dari *internet lamp*.

Aktor: *User*

Tabel 3-13 *Use Case Secenario Input Serial Number*

Aktor	Reaksi Sistem
Sekenario Normal	
5. Aktor login pada aplikasi	
	6. Menampilkan halaman utama.
7. Memilih *Add Device*	
	4. Menambah device *Internet lamp*
5. Memasukan *serial number internet lamp*	
	6. Validasi *serial number internet lamp*
	7. Masuk ke halaman kontrol *internet lamp*
Sekenario Alternatif	
1. Aktor login pada aplikasi	
	2. Menampilkan halaman utama.
3. Memilih *Add Device*	
	4. Menambah device *internet lamp*
5. Masukan *serial number internet lamp*	
	6. Validasi *serial number internet lamp*

	7 Menampilkan pesan *serial number invalid*, masuk ke halaman kontrol *internet lamp* gagal
8. Masukan *seria number* yang *valid*	
	9. Validasi *serial number internet lamp*
	10. Masuk ke halaman kontrol *internet lamp*

9) Nama *Use Case*: Control *Internet lamp*

Deskripsi: Menjalankan proses untuk mengontrol *internet lamp.*

Aktor: *User*

Tabel 3-14 *Use Case Secenario kontrol Internet lamp*

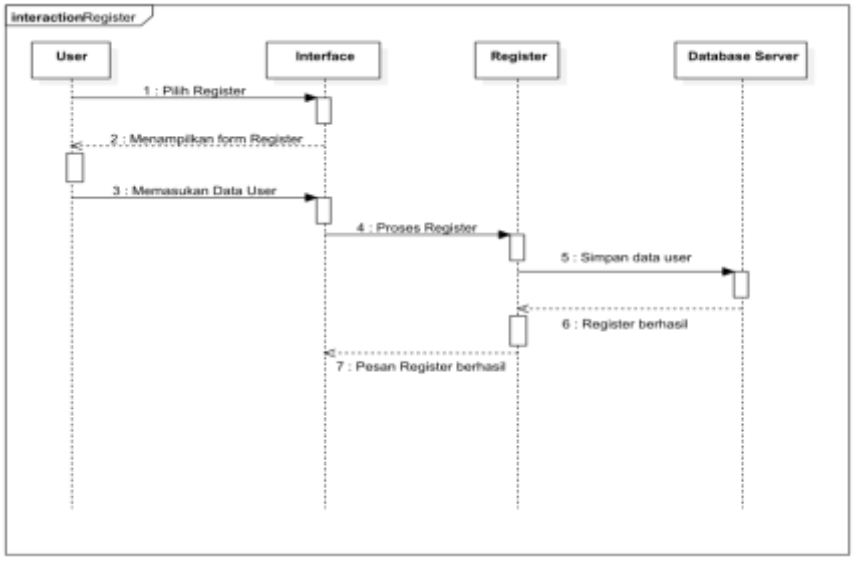
Aktor	Reaksi Sistem
Sekenario Normal	
1. Aktor login pada aplikasi	
	2. Menampilkan halaman utama.
3. Memilih *Add Device*	
	4. Menambah device *internet lamp*
5. Memasukan *serial number internet lamp* yang valid	
	6. Validasi *serial number internet lamp*
	7. Masuk ke halaman kontrol *internet lamp*
8. Tekan *Refresh*	

	9. Cek *device internet lamp* aktif atau tidak
10. Tekan *On*	
	11. Proses menyalakan lampu
12. Tekan Off	
	13. Proses mematikan lampu
14. Tekan *Back*	
	15. Proses Kembali ke halaman utama.

3.2.6.2 *Sequence* Diagram

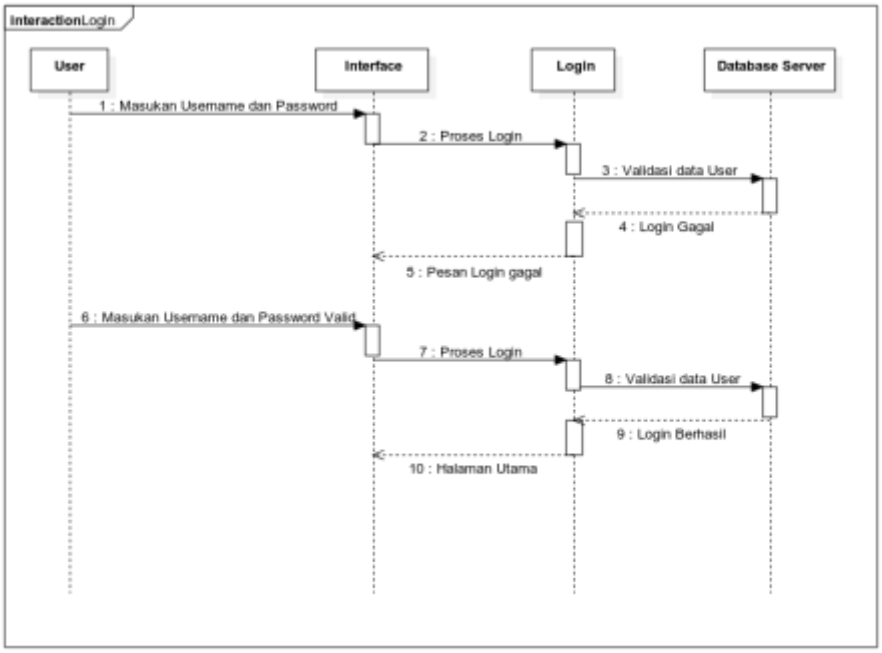
Sequence diagram menggambarkan aliran kerja objek pada *use case* dengan mendeskripsikan waktu hidup dan pesan yang dikirimkan dan diterima antar objek[13]. *Sequence* diagram pada sistem ini sebagai berikut:

1. *Sequence* Didagram *Register.*



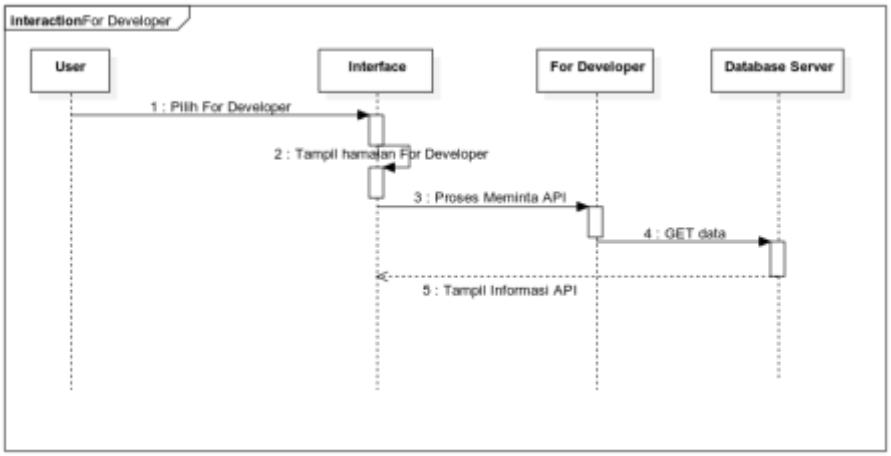
Gambar 3-15 *Sequence* Didagram *Register.*

2. *Sequence* Diagram *Login*



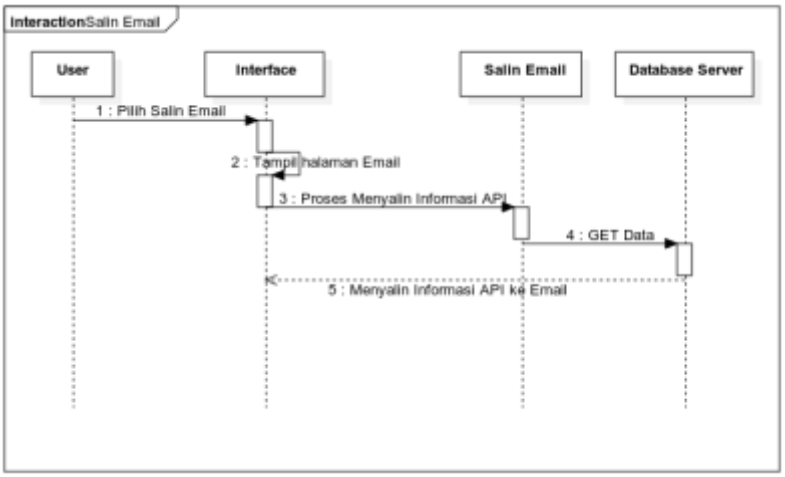
Gambar 3-16 *Sequence* Didagram *Login.*

3. *Sequence* Didagram *For Developer*



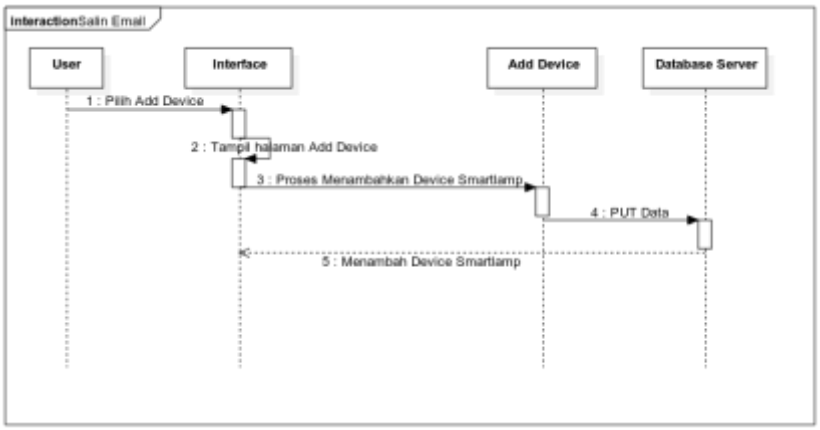
Gambar 3-17 *Sequence* Didagram *For Developer*

4. *Sequence* Didagram Salin *Email*



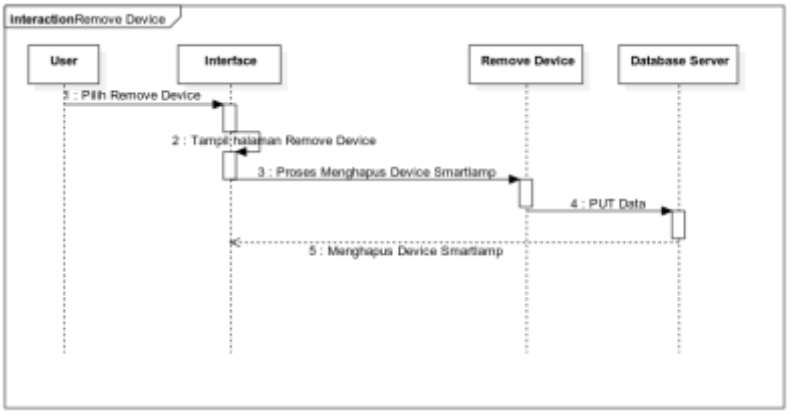
Gambar 3-18 *Sequence* Didagram Salin Email

5. *Sequence* Didagram *Add Device*



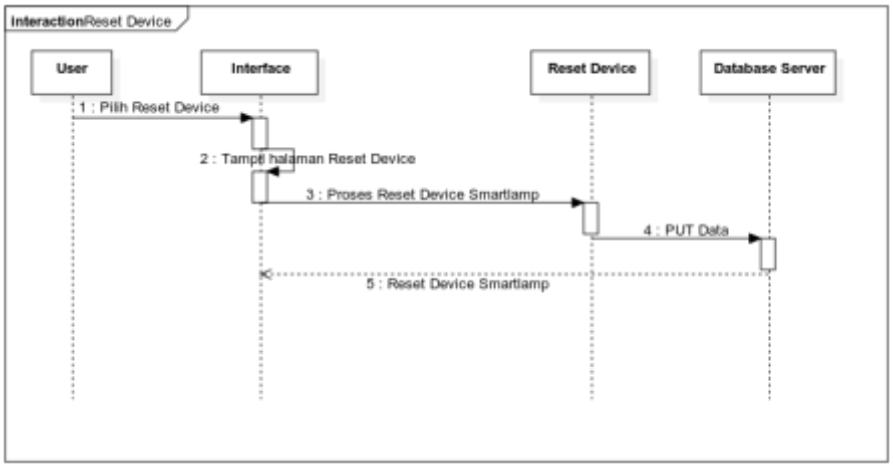
Gambar 3-19 *Sequence* Didagram Add Device

6. *Sequence* Didagram *Remove Device*



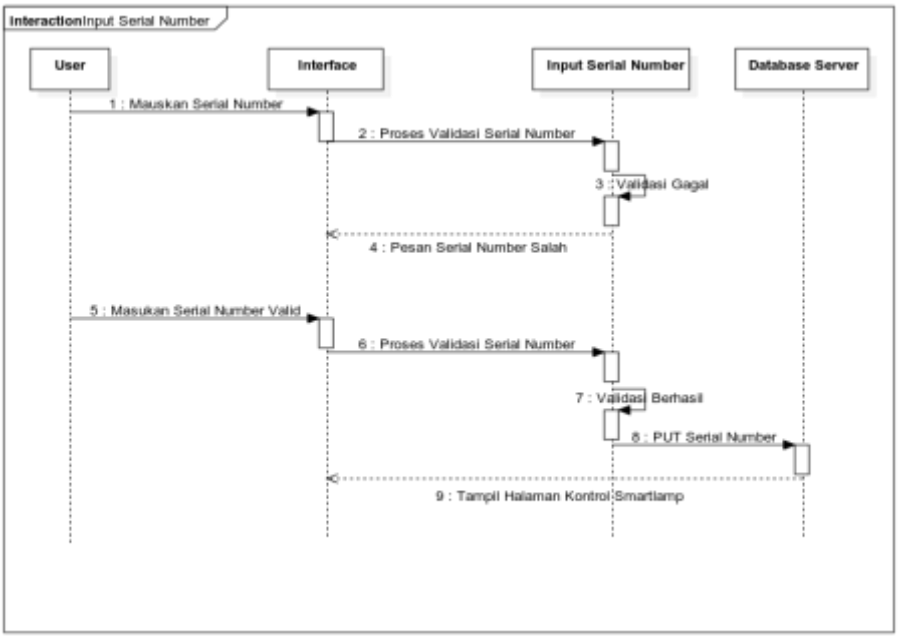
Gambar 3-20 *Sequence* Didagram *Remove Device*

7. *Sequence* Didagram *Reset Device*



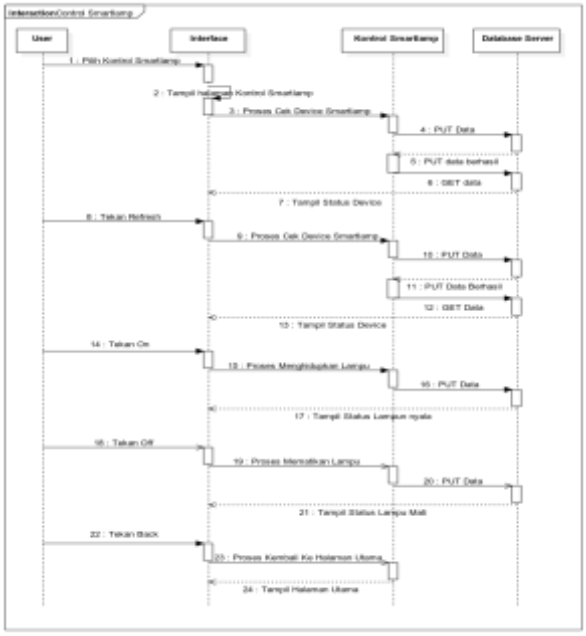
Gambar 3-21 *Sequence* Didagram *Reset Device*

8. *Sequence* Didagram *Input Serial Number*



Gambar 3-22 *Sequence* Didagram *Input Serial Number*

9. *Sequence* Didagram *Control Internet lamp*



Gambar 3-23 *Sequence* Didagram Kontrol *Internet lamp*

3.2.6.3 *Class* Diagram

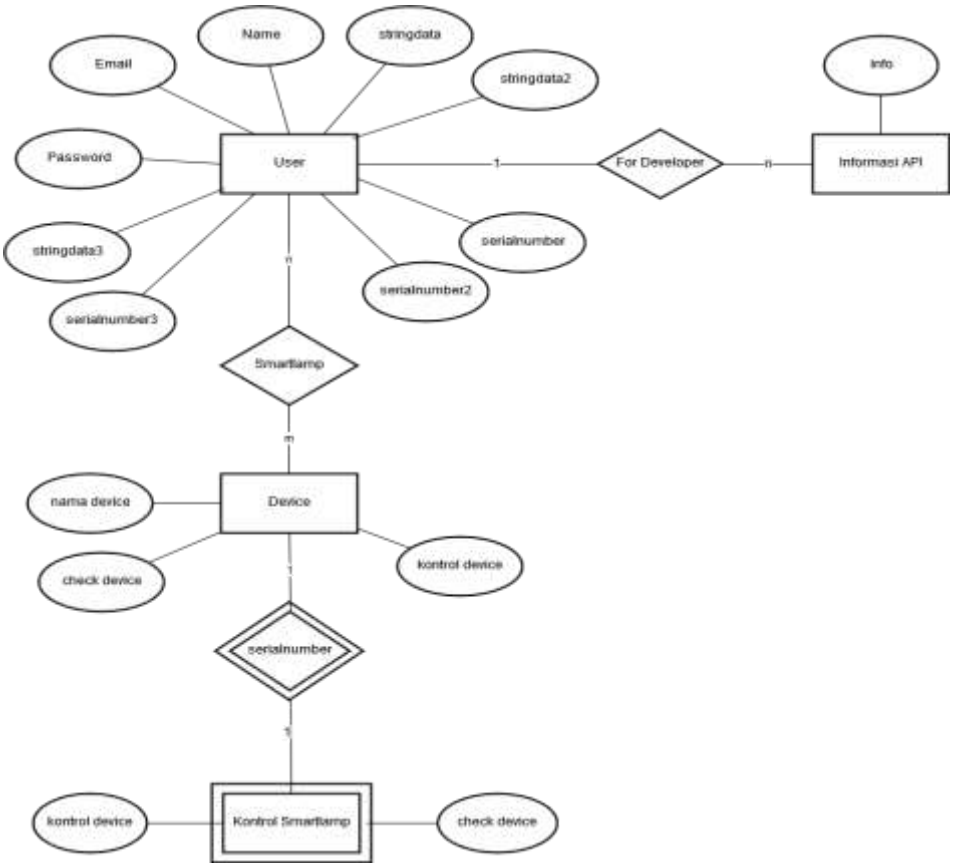
Class diagram adalah diagram yang digunakan untuk menampilkan beberapa kelas yang ada dalam perangkat lunak yang sedang dikembangkan. Class diagram menggambarkan struktur dan deskripsi kelas beserta hubungan kelas yang satu dengan yang lain[13]. Berikut adalah *Class* Diagram dari aplikasi layanan IoT yang terlihat pada gambar 3-24.



Gambar 3-24 *Class Diagram* sistem

3.2.6.4 Analisis Basis Data

Dalam memodelkan data dan menggambarakan hubungan antara data yang ada pada sistem digunakan alat bantu yaitu *Entry Relationship Diagram* (ERD)[13]. Sistem yang akan dibangun dapat dilihat hubungan antar entitas dan atributnya. Sistem yang akan dibangun memiliki ERD seperti pada gambar 3-25.



Gambar 3-25 *Entry Relationship Diagram* (ERD) sistem

Untuk keterangan mengenai entitas dan atribut pada Gambar 3-25 dapat dilihat pada Tabel 3-15 berikut ini :

Tabel 3-15 *Entitas* dan *Atribute*

No.	Entitas	Atribut
1.	User	Name, Email, Password, stringdata, stringdata, stringdata2, stringdata3, serialnumber, serialnumber2, serialnumber3
2.	Informasi API	info

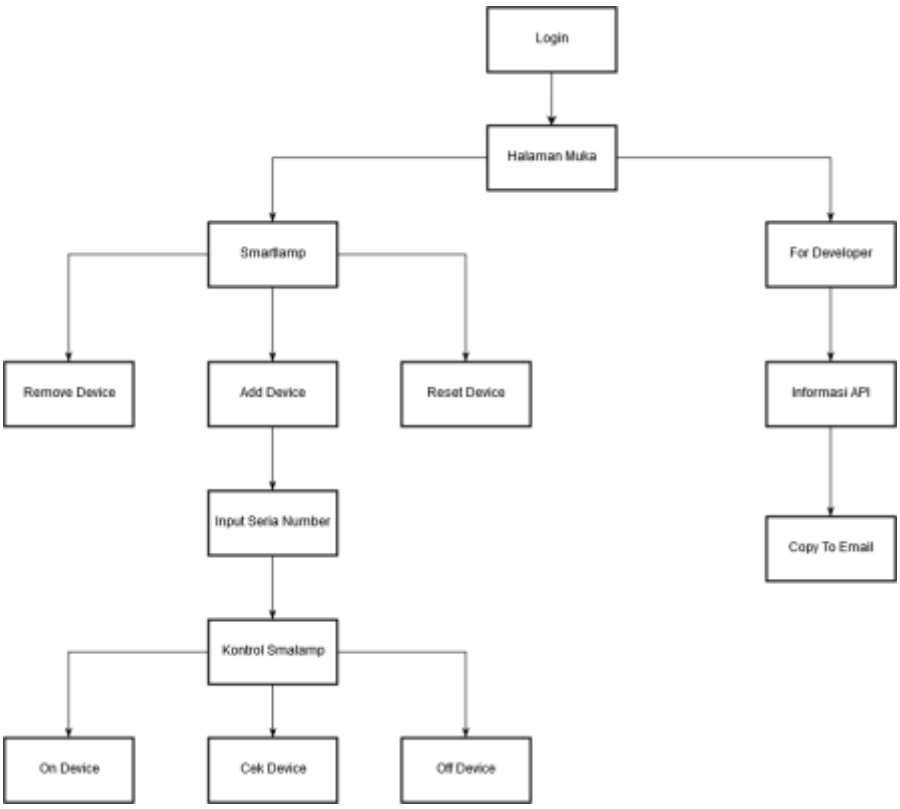
3.	Device	Nama device, check device, kontrol device
4.	Serial Number	serialnumber, serialnumber2, serialnumber3
5.	Kontrol *Internet lamp*	check device, kontrol device.

3.2.7 Perancangan Sistem Pada Perangkat Lunak Android

Dalam membangun aplikasi, salah satu hal yang harus dilakukan yaitu perancangan sistem. Perancangan sistem pada perangkat lunak terdiri dari arsitektur menu dan perancangan antarmuka.

3.2.7.1 Arsitektur Menu

Dalam perancangan sebuah program, dibutuhkan suatu struktur menu yang menjelaskan mengenai menu yang akan dibangun. Gambar 3-26 menunjukan arsitekur menu dari perangkat lunak Layanan IoT.

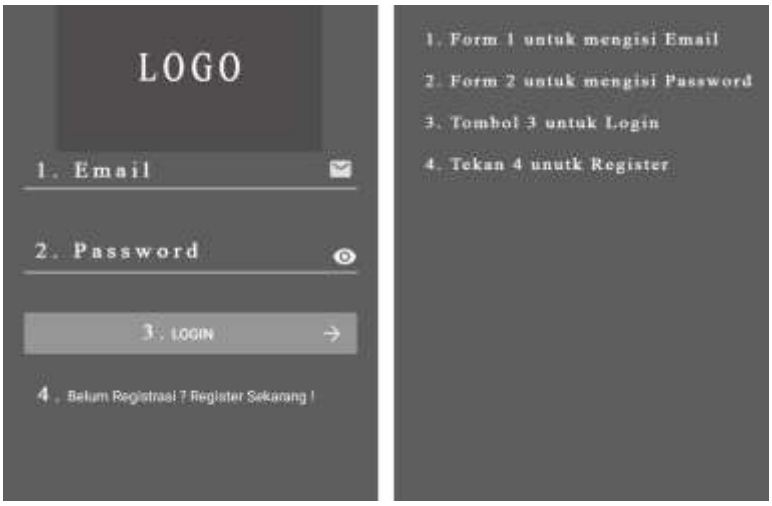


Gambar 3-26 Arsitektur Menu perangkat lunak Layanan IoT

3.2.7.2 Perancangan Antarmuka

Antarmuka merupakan suatu media komunikasi antara pengguna dengan aplikasi yang dibangun. Oleh karena itu dalam perancangan sebuah aplikasi dibutuhkan suatu perancangan antarmuka guna memberikan gambaran umum bagi pengguna. Perancangan antarmuka aplikasi ini dibagi menjadi beberapa bagian, diantaranya :

1. Perancangan Antarmuka Halaman Login.



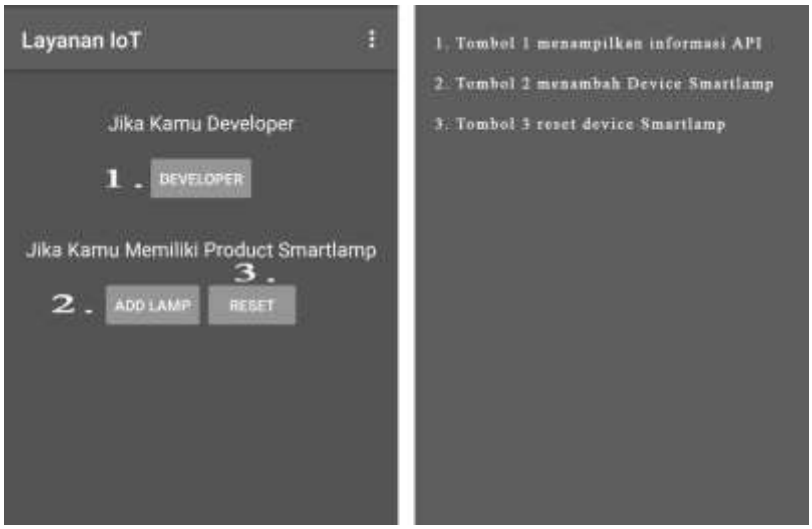
Gambar 3-27 Perancangan Antarmuka Halaman *Login*

2. Perancangan Antarmuka Halaman Register



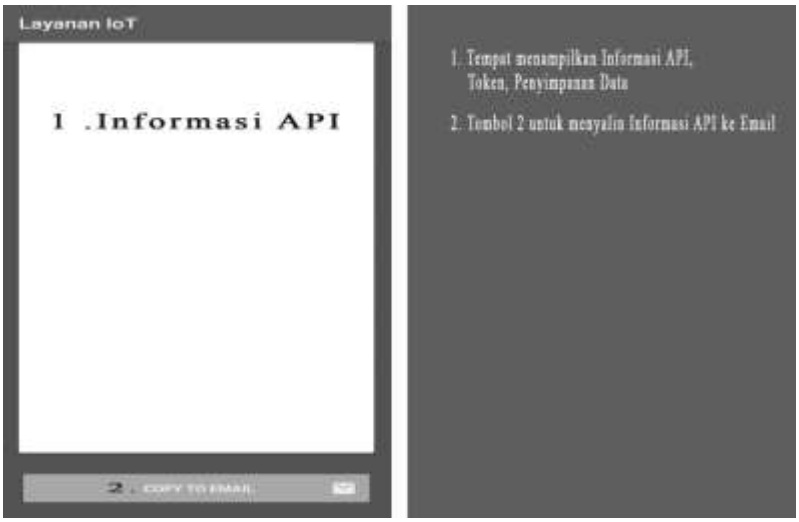
Gambar 3-28 Perancangan Antarmuka Halaman *Register*

3. Perancangan Antarmuka Halaman Muka



Gambar 3-29 Perancangan Antarmuka Halaman Muka

4. Perancangan Antarmuka Halaman Informasi API.



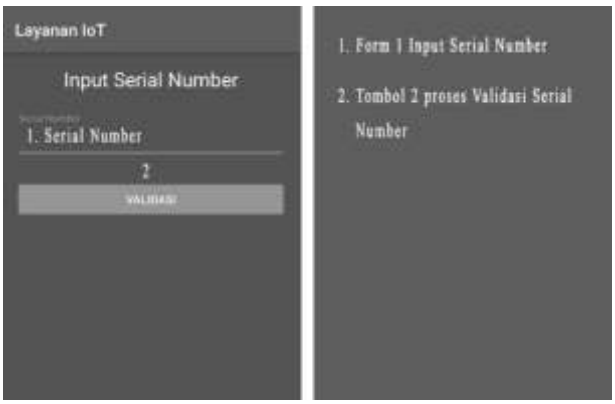
Gambar 3-30 Perancangan Antarmuka Halaman Informasi API

5. Perancangan Antarmuka *Add, Remove, Reset Device Internet lamp*.



Gambar 3-31 Perancangan Antarmuka *Add, Remove, Reset Device Internet lamp*

6. Perancangan Antarmuka Halaman *Input* Serial Number



Gambar 3-32 Perancangan Antarmuka Halaman *Input* Serial Number

7. Perancangan Antarmuka Halaman Kontrol *Internet lamp*

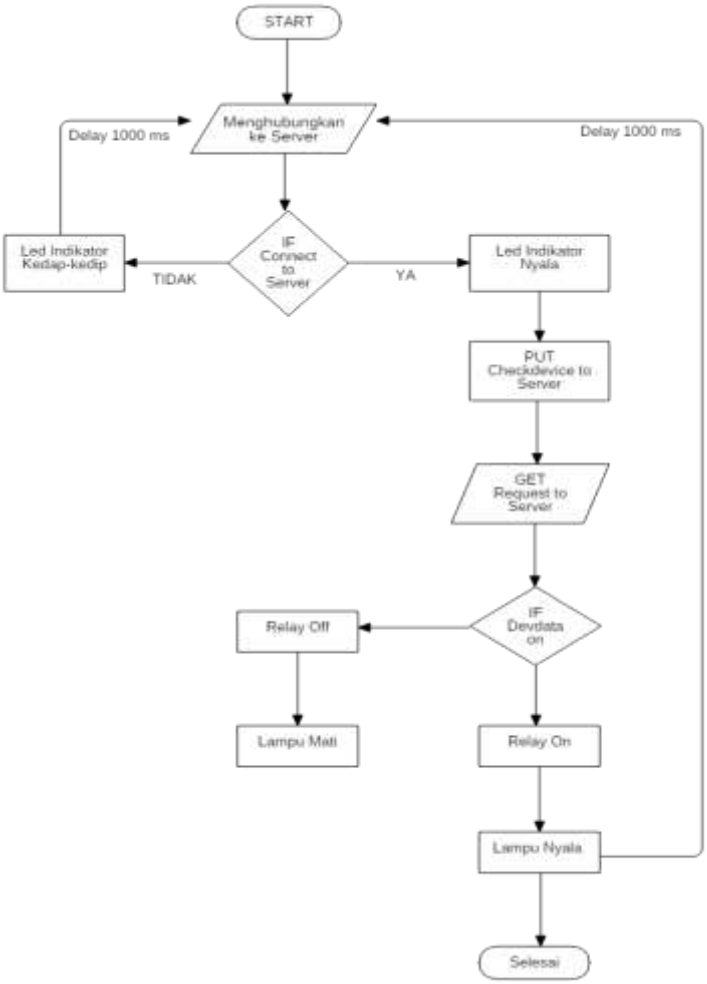


Gambar 3-33 Perancangan Antarmuka Halaman Kontrol *Internet lamp*

3.2.8 Konfigurasi Program ESP8266

ESP8266 dikonfigurasi menggunakan *software* Arduino IDE dan ada beberapa langkah yang harus diperhatikan yaitu sebagai berikut:

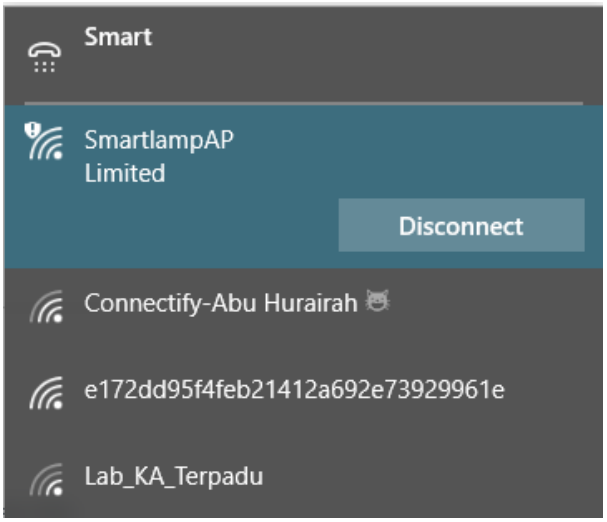
1. Sambungkan ESP8266 ke Leptop atau Komputer yang sudah terinstall Arduino IDE.
2. Berikut adalah *Flowchart* program yang di *upload* pada ESP8266[14]:



Gambar 3-34 *Flowchart* diagram program *Internet lamp*

3. Setelah berhasil *upload* program selanjutnya test konfigurasi ESP8266.

4. Hidupkan ESP8266 dan cek di leptop atau *smartphone* apakah terdeteksi akses point yang bernama SmartlampAP lalu hubungkan. Apabila terhubung berarti konfigurasi ESP8266 berjalan dengan baik sebagai *wifi manager*.



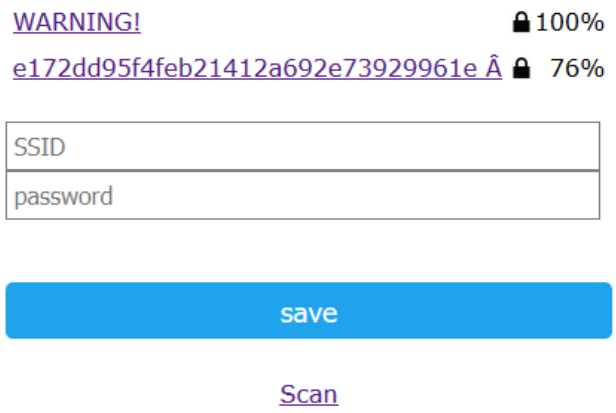
Gambar 3-35 Akses Point ESP8266 Terdeteksi

5. Setelah terhubung ke SmartlampAP masuk ke halaman awal *wifi manager*, akan terlihat sebagai berikut.



Gambar 3-36 Halaman awal *wifi manager* ESP8266

6. Pilih menu Configurasi WiFi yang berfungsi untuk menghubungkan ESP8266 ke wifi yang berada di sekitarnya yang terhubung ke internet.



Gambar 3-37 Halaman untuk menghubungkan ESP8266 ke wifi di sekitarnya

7. Disana terlihat beberapa wifi yang aktif pilih salah satunya. Masukan SSID dan password wifi lalu pilih save.
8. Konfgurasi ESP8266 telah berhasil sudah terhubung ke internet dan lampu dapat di kontrol melalui *smartphone*.