BAB III

PELAKSANAAN KERJA MAGANG

3.1. Kedudukan dan Koordinasi

Pada kerja magang ini, perancangan sistem absensi dengan pengenalan wajah perusahaan dibantu oleh Bapak Erick Alviyendra sebagai IT *manager*.

3.2. Tugas Yang Dilakukan

Dalam pelaksanaan kerja magang, pembuatan aplikasi dilaksanakan secara bertahap, berikut hal yang dilakukan setiap minggunya:

Tabel 3.1 Tugas Yang Dilakukan

Minggu	Tugas Yang dilakukan
1	-Mendapatkan *user requirements* -Mengerjakan di Python 3.7 -Melakukan compile terhadap Dlib dan OpenCV berbasis Python memakai Nvidia® CUDA™
2	-Memindahkan program ke bahasa pemrograman C++ -Melakukan compile terhadap Dlib dan OpenCV berbasis C++ memakai Nvidia® CUDA™
3	- Menyesuaikan *tolerance rate* dari model *pre-trained* Dlib.

Tabel 3.1 Tugas Yang Dilakukan (lanjutan)

Minggu	Tugas Yang dilakukan
3	-Merancang skema interaksi *client* dan *server*
4.	-Mengimplementasi interaksi *client* dan *server* - Melakukan compile ulang Dlib dan OpenCV memakai Nvidia® CUDA™
5	-Mengimplementasikan sistem penyimpanan database
6	-Memasang *library* Dirent -Mengimplementasikan *refresh timer* -Mengintegrasikan 7-zip
7	-Membuat jembatan bagi Python code (exec) -Mengimplementasikan *face liveness* yang menggunakan pergerakan mata. -Melakukan presentasi ke IT Director
8	-Melakukan *recompile* Dlib dan OpenCV tidak memakai Nvidia® CUDA™ -mengimplementasi *multi-threading* pada *client*. -Membuang integrasi 7-Zip yang sudah dipasang -Membuang implementasi *face liveness*
9	-Membuat prototype *user interface* -Membuat API FLASK untuk koneksi server dan client
10	-Memasang aplikasi di tablet Intel® Atom™ yang disediakan user

Tabel 3.1 Tugas Yang Dilakukan (lanjutan)

Minggu	Tugas Yang dilakukan
11	-Membuat kode python untuk database dan query return ke client
12	-Menambah *feedback text* di client dan mengupdate dependency
13	-Menambah fitur tambahan
14	-Memperbaiki stabilisasi aplikasi -Membuat aplikasi siap di deploy dengan *Copy and Run*
15	-Mengoptimisasi Aplikasi
16	-Melakukan *quality check* terhadap kualitas aplikasi *prototype*

Hal pertama yang dilakukan di minggu pertama adalah mendapatkan *user requirements* dari Bapak Erick Alviyendra. Setelah mendapat *user requirements*, yang dilakukan adalah melakukan *compiling* terhadap OpenCV dan Dlib, compile berbasis Python 3.7 dan menggunakan Nvidia® CUDA™ 10.1, Intel SSE 4.2. Setelah selesai compiling, hal yang dilakukan adalah membuat *face recognition runtime module* berbasis Python 3.7 dengan mengintegrasikan OpenCV dan Dlib yang sudah di compile sebelumnya. Setelah dilakukan pengujian awal, sayangnya hasil performancenya tidak memuaskan karena python interpreter.

Minggu kedua merupakan titik start perpindahan dari bahasa Python 3.7 ke C++, hal pertama yang dilakukan di minggu kedua adalah melakukan *compiling* terhadap OpenCV dan Dlib, compile berbasis C++ 2019 dan menggunakan Nvidia®

CUDA™ 10.1, Intel SSE 4.2. Setelah sukses compiling, hal selanjutnya yang dilakukan adalah membuat *face recognition runtime module* berbasis C++ 2019 dengan mengintegrasikan OpenCV dan Dlib yang sudah di compile sebelumnya. Setelah dilakukan pengujian awal, hasilnya cukup baik untuk dapat diperbaiki dan dikembangkan lebih lagi.

Di minggu ketiga dilakukan stress test terhadap aplikasi, dengan hasil banyaknya *false detection* di dalam *face recognition module,* sehingga dilakukan penyesuaian *tolerance rate* dari model *pre-trained* Dlib agar dapat akurat dengan user. Lalu setelah program pendeteksi mukanya sudah selesai, dilakukan perancangan skema interaksi *client* dan *server* berdasarkan apa yang diinginkan IT *manager*. Rancangannya adalah database local dengan guest mode (tidak dapat melakukan write terhadap database local tersebut), kalkulasi di client dan server memasukan ke dalam database.

Di minggu keempat, hal yang dilakukan adalah mulai membangun sistem interaksi *client* dan *server* yang sudah dirancang di minggu ketiga. Selain itu, juga dilakukan pemasangan Intel® MKL, melakukan *recompile* terhadap Dlib menggunakan Intel® TBB dan OpenCV menggunakan OpenMP. Di minggu kelima, hal yang dilakukan adalah mengimplementasikan sistem penyimpanan database foto karyawan dalam bentuk *serialized file*. *Serialized file* ini sendiri berekstensi ".facedat" dan menggunakan algoritma yang telah tersedia di Dlib.

Hal yang dilakukan di minggu keenam adalah memasang *library* Dirent untuk memudahkan pembacaan file didalam *subfolder* di semua aplikasi. Selain itu,

mengimplementasikan *timer* untuk melakukan *refresh database* local di *client* dari *server*. Terakhir, mengintegrasikan 7-zip untuk melakukan MD5 checksum dalam pengiriman data dari *server* ke *client (*masih simulasi *same device).*

Di minggu ketujuh, hal yang dilakukan adalah pemasangan jembatan bagi kode Python yang akan dibuat perusahaan nantinya agar dapat dieksekusi dan dibaca hasilnya di C++, lalu membuat *face liveness detection* berbasis pendeteksian mata agar dapat membedakan foto di kartu karyawan dan wajah karyawan asli di kamera,

Setelah fiturnya dirasa cukup, melakukan presentasi ke IT Director untuk menanyakan kesesuaian aplikasi dengan kebutuhan perusahaan, dan setelah presentasi IT Director ingin beberapa perubahan, perubahan pertama adalah semua verifikasi muka terjadi di *server* dan bukan di client, tidak ada *database* lokal. Selain hal diatas, IT Director ingin program yang cukup untuk berjalan di perangkat yang sudah ada di toko, sehingga Nvidia® CUDA™ harus dibuang. Karena perbedaan source code menggunakan dan tidak menggunakan Nvidia® CUDA™ hanya sedikit, Nvidia® CUDA™ (x64) tetap di *maintain* di *branch* terpisah, ditambah dengan dua *branch* baru yaitu x86 dan x64. Server menggunakan x64 sementara client menggunakan x86 karena perangkat hanya memiliki OS Windows 10 x86 dan 2 GB RAM

Di minggu kedelapan, karena tidak memakai Nvidia CUDA™ dan murni memakai Intel® Atom™ maka banyak fitur harus dibuang, termasuk *face liveness* detection mata yang minggu lalu dipasang karena terlalu berat di Intel® Atom™. Karena terjadi perubahan requirement maka integrasi 7-Zip juga ikut dibuang. Setelah

membuang hal-hal yang tidak diinginkan IT Director, dilakukan penyesuaian ulang aplikasi dan memindahkan kerja verifikasi wajah ke server. Setelah sistem berjalan sesuai kemauan IT Director, dilakukan implementasi *multithreading* untuk mempercepat pekerjaan program dan mengefisiensikan waktu.

Di minggu kesembilan, atas permintaan *user*, di minggu ini membuat prototype *user interface* yang dengan keterangan *user* bahwa *user interface* disini tidak akan dipakai di kemudian hari karena ada tim *graphic designer* yang akan membuatkan ulang tampilan aplikasi yang dibuat. Membuat API FLASK Python yang berfungsi sebagai *service* beserta *script* untuk koneksi *client* ke *server* sekaligus.

Di minggu kesepuluh, hal yang dilakukan adalah memasang aplikasi di tablet Intel® Atom™ yang disediakan user dan memasang *dependency*-nya. Di minggu kesebelas, hal yang dilakukan adalah membuat *script* python untuk database, dan sekaligus mengganti query ke client dari server yang tadinya bersifat *spontaneous* menjadi delayed *query* yang sudah di cek validitasnya.

Di minggu keduabelas, hal yang dilakukan adalah menambah jumlah textbox di client dari 5 ke 10 dan mengupdate library Dlib dan melakukan *recompiling* terhadap aplikasi yang telah dibuat. Di minggu ketigabelas, hal yang dilakukan adalah memasang fitur-fitur tambahan yang bersifat *nice-to-have*, walau tidak wajib dan tidak ada di *requirement*, tetapi fitur-fitur ini cukup membantu untuk perkembangan program di masa depan. Contoh fiturnya adalah *offline backup request copy* yang ada saat *client* gagal mengirim *request* ke *server*. *Request* yang gagal ini ada di dalam file

yang bernomor keatas hingga 2 juta, membuka peluang untuk membuat *auto upload* terhadap file yang gagal nantinya

Di minggu keempatbelas, aplikasi di perbaiki stabilitasnya, dan membuat aplikasi siap di *deploy* di device manapun dengan *Copy and Run.* Di minggu kelima belas, yang dilakukan hanyalah mengoptimisasi aplikasi untuk dapat bekerja di Intel® Atom™ secara optimal. Di minggu terakhir, hal yang dilakukan adalah melakukan *quality check* terhadap kualitas aplikasi yang telah memasuki *final prototype.*

3.3. Uraian Pelaksanaan Kerja Magang

3.3.1. Proses Pelaksanaan

A. Perancangan dan Desain Sistem

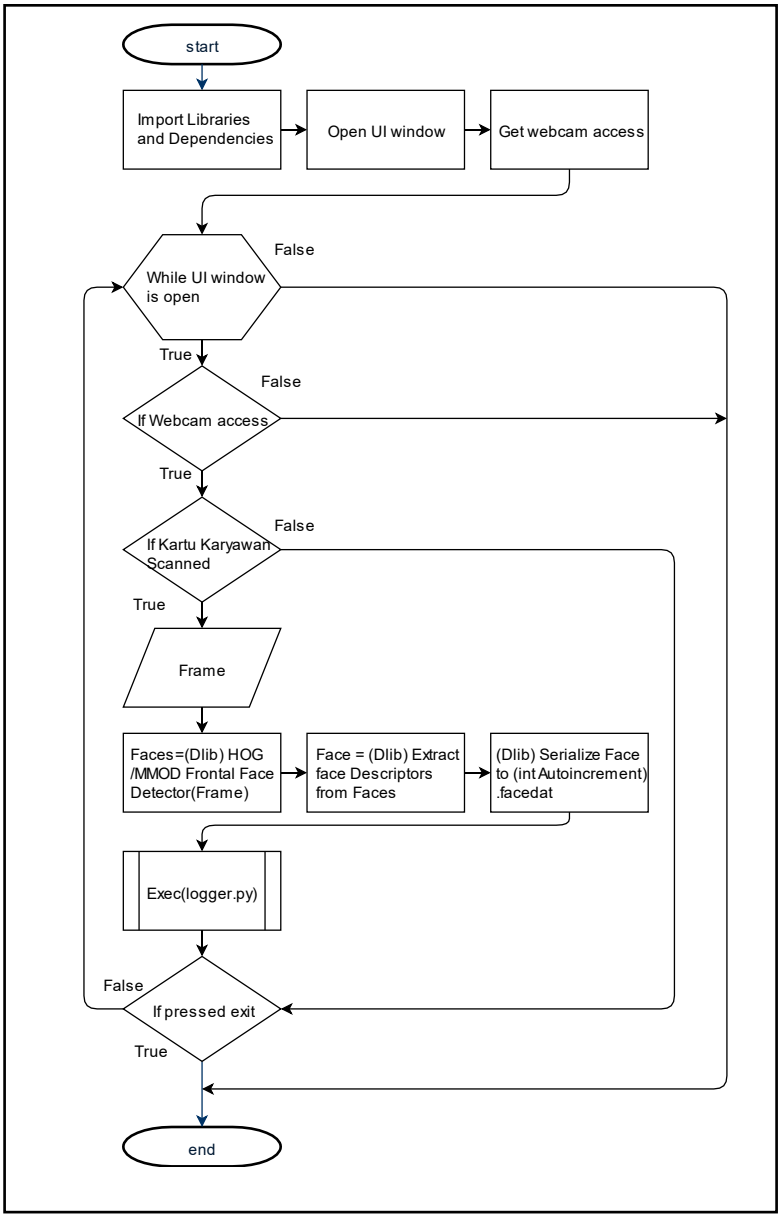
Pada awalnya, *user* memiliki *requirement* sebagai berikut:

- Pengenalan wajah berbasis kamera, dapat mengenali wajah karyawan
- Berbahasa Python

Pada perkembangannya, *user requirement* berubah sehingga sebagai berikut:

- Memiliki fungsi client-server, dimana server merupakan tempat verifikasi
- Program inti dalam C++, sementara sistem koneksi, database, dan yang lain-lain dikerjakan di Python
- Berjalan dengan baik di tablet Windows 10 64 bit processor Intel® Atom™ Z3560 (quad-core 1.83 ghz), RAM 4GB

Berdasarkan user requirement tersebut, ada 3 program yang akan dibuat, yaitu *client, server* dan *server database*. Masing-masing program memiliki *flowchart*-nya masing masing, dan server database memiliki 3 anak *flowchart*. Semua *flowchart* ini akan dijelaskan beserta gambar *flowchart*-nya dibawah ini.



Gambar 3.1 Flowchart Program Client

Dari flowchart diatas, mengambarkan alur kerja program *client* yang tersebar di berbagai daerah. “Import Libraries and Dependencies” adalah melakukan import terhadap library OpenCV, Dlib, dan Dirent, beserta file-file yang dibutuhkan untuk runtime. Untuk mencegah program mengalami *crash* saat tidak menemukan foldernya, program otomatis men-*generate* folder yang dibutuhkan untuk membaca atau menulis file.

User ingin bahwa wajah dibandingkan di *server* agar tidak dapat di manipulasi oleh *client*. Karena harus di transfer datanya ke *server* untuk dibandingkan, tidak efisien secara *bandwidth* bagi *client* untuk mengirim *webcam footage* mentah ke *server*, jadi solusinya adalah membuat *client* melakukan pengenalan wajah di pc *client*, data wajahnya yang dikirim ke *server* untuk di proses lebih lanjut.

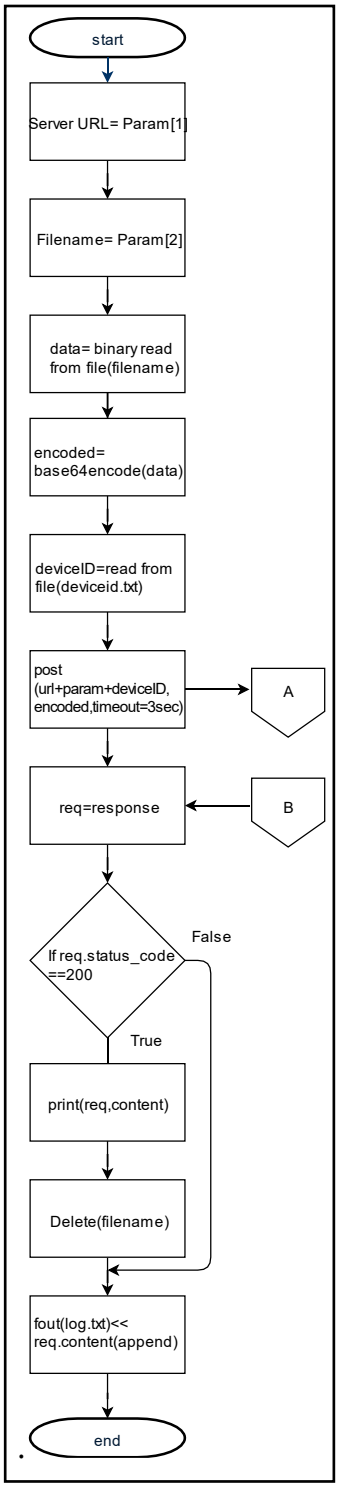
Berhubung proses *neural network* memakan *resource* yang cukup banyak, agar tidak menganggu *main thread* untuk *user video feedback*, semua proses *neural network* Dlib dilakukan di *thread* yang berbeda dari *main thread*. Main thread akan membuat satu *Thread* HOG Face Detector tambahan yang akan beraksi saat ada scan barcode untuk kartu karyawan, sekali lalu tidur sedetik (sleep 1000ms) sebelum mengulang dirinya sendiri sembilan kali lagi agar tidak membuat *bottleneck* di processor Intel® Atom™. *Thread* HOG Face Detector yang sudah selesai mengekstrak semua wajah dari satu *frame* akan memanggil *thread* baru yang bertugas untuk melakukan ekstraksi *face descriptor* wajah, melakukan *serialization* kedalam suatu file, dan memanggil Python untuk meng-*upload* filenya ke *server* untuk verifikasi wajah.

HOG Face Detector adalah metode pendeteksi wajah dari Dlib yang paling cepat dijalankan di CPU, hanya dapat mendeteksi wajah yang menghadap depan. Proses ini memakai model *neural network* yang sudah dilatih (*pre-trained)* shape_predictor_5_face_landmark.dat yang telah disediakan dari *library* Dlib itu sendiri.

Untuk *build* NVIDIA CUDA, pendeteksi wajah menggunakan MMOD Face Detector yang disediakan Dlib. MMOD Face Detector sendiri adalah metode pendeteksi wajah yang paling cepat dijalankan di GPU. Berhubung Alfamart menggunakan tablet Intel® Atom™, maka *build* NVIDIA CUDA ada hanya sebagai *future proof build* bila di kemudian hari perangkat menggunakan NVIDIA CUDA. Open UI Window membuka *user interface prototype* yang menampilkan *video webcam* disertai dengan tempat untuk menerima feedback beserta *real time clock.*

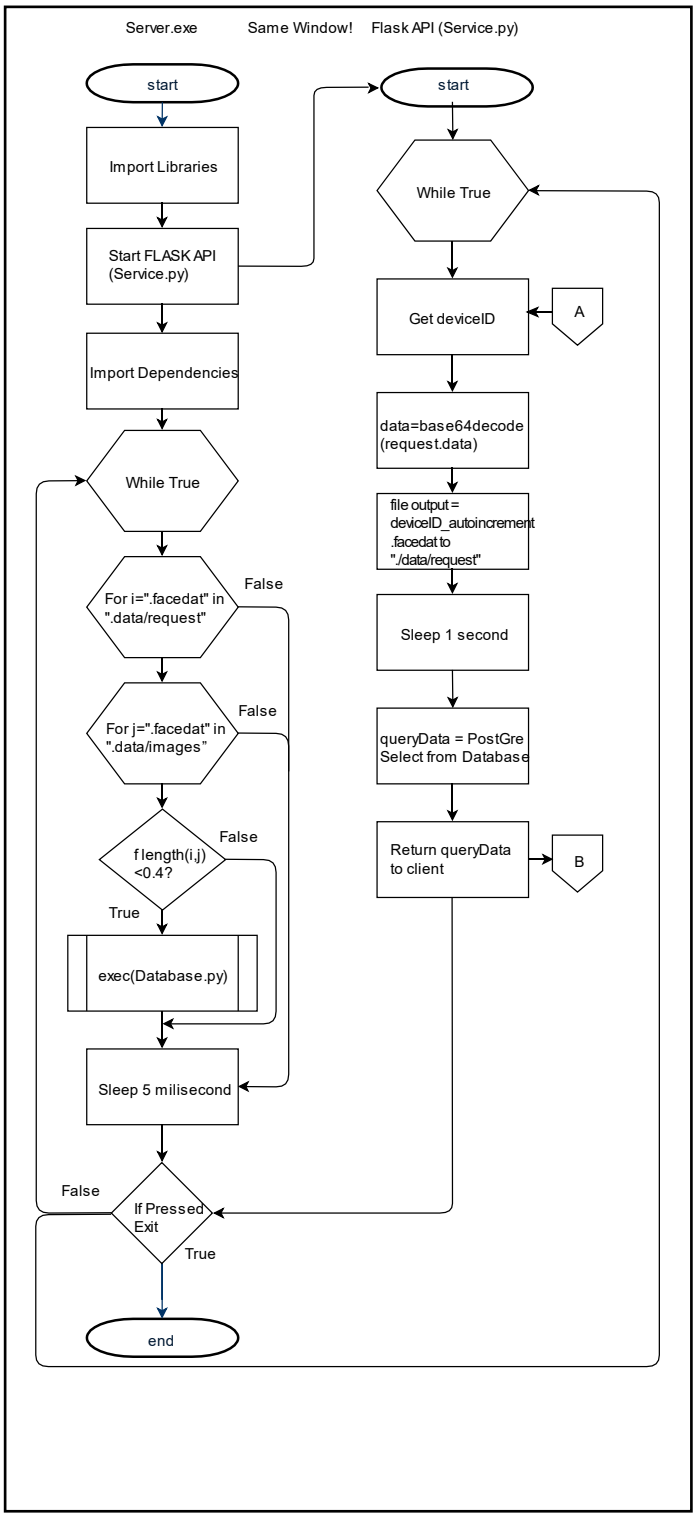
"Extract Face Descriptor from Faces" adalah proses mengekstrak 128 indikator pengukuran wajah menggunakan model *neural network* yang sudah dilatih (*pre-trained)* dlib_face_recognition_resnet_model_v1.dat yang telah disediakan dari library Dlib itu sendiri.

"Serialize Face to (Int AutoIncrement).facedat" setelah 128 indikator pengukuran wajah telah di extrak, data tersebut di *serialize* kedalam suatu file berekstensi ".facedat" yang algoritmanya disediakan oleh Dlib. Nama filenya menghitung *Integer* naik keatas dari 0 sampai 2 juta, setelah 2 juta kembali lagi ke 0. File akan berada di ".data/clientrequest". Exec(logger.py) bertugas memanggil kode Python untuk melakukan upload file ini ke server dan meminta feedback.



Gambar 3.2 Flowchart logger.py

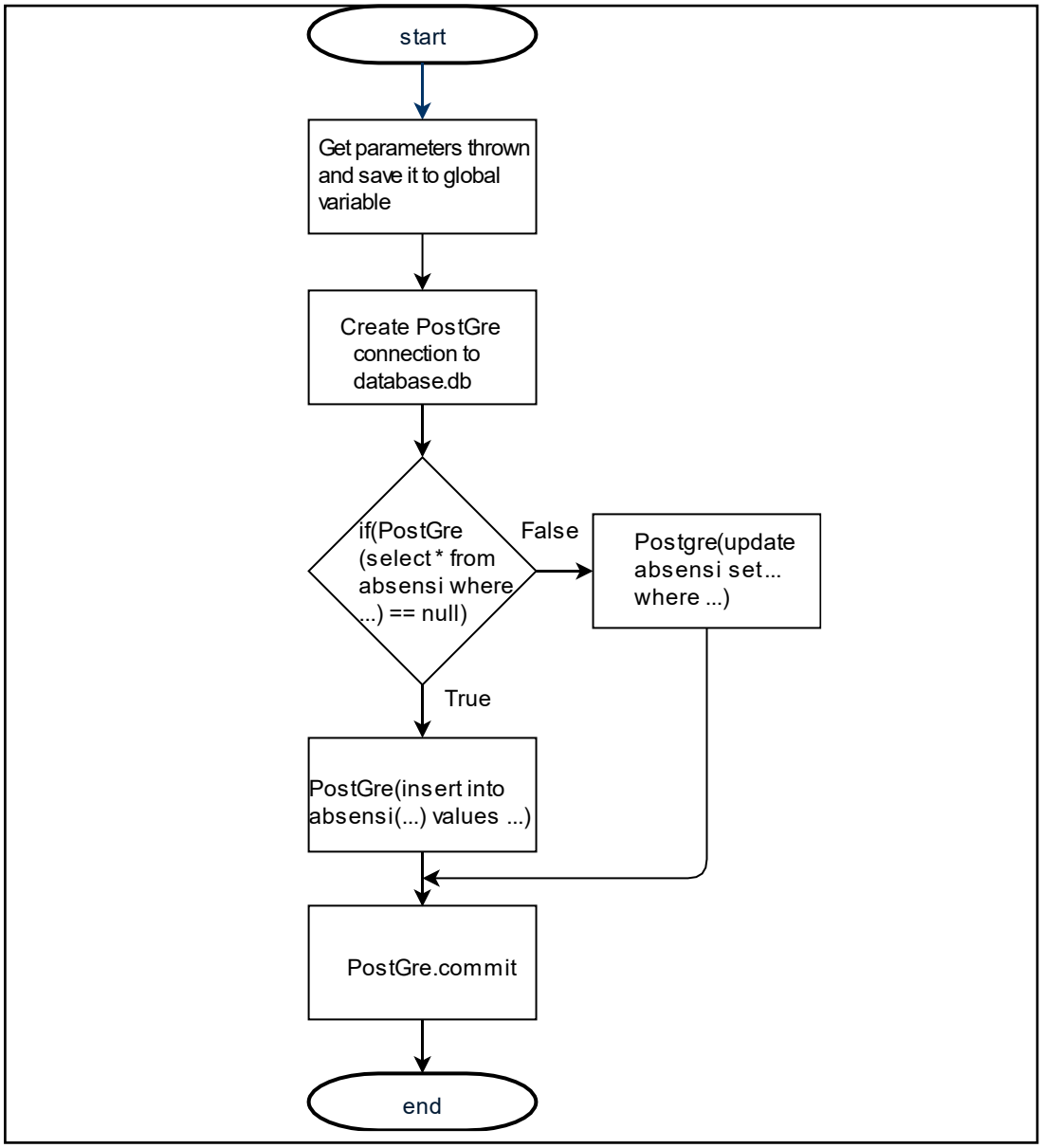
Flowchart diatas menggambarkan alur *script Python* logger.py yang dipanggil oleh C++. Pertama Python akan mengambil *server* url dan filename dari *parameter* 1 dan 2, lalu file dicari dengan nama file tersebut dan dibaca *binary*-nya. Setelah membaca file binary, deviceID akan dibaca dari deviceID.txt lalu ditambah dengan *parameter* di Python dan deviceID tersebut dikirim ke server melalui request.post dengan *timeout* 3 detik. Bila *request code* hasil *response* dari *server* hasilnya sama dengan 200, file yang dibaca tersebut di delete dan *response message* nya di cetak. Setelah itu hasil *response* di masukkan kedalam *log file*. Bila *timeout*, maka program akan terus berjalan sebagaimana mestinya dan menulis *error timeout* ke *log file* tersebut.



Gambar 3.3 Flowchart Program Server

Dari *flowchart* diatas, mengambarkan alur kerja program *server* yang terletak di kantor pusat. “Import Libraries” adalah melakukan *import* terhadap library OpenCV, Dlib, dan Dirent. Exec(Python) bertugas memanggil kode Python untuk memberi *feedback* ke *client* dan melakukan *update* ke *database* absensi

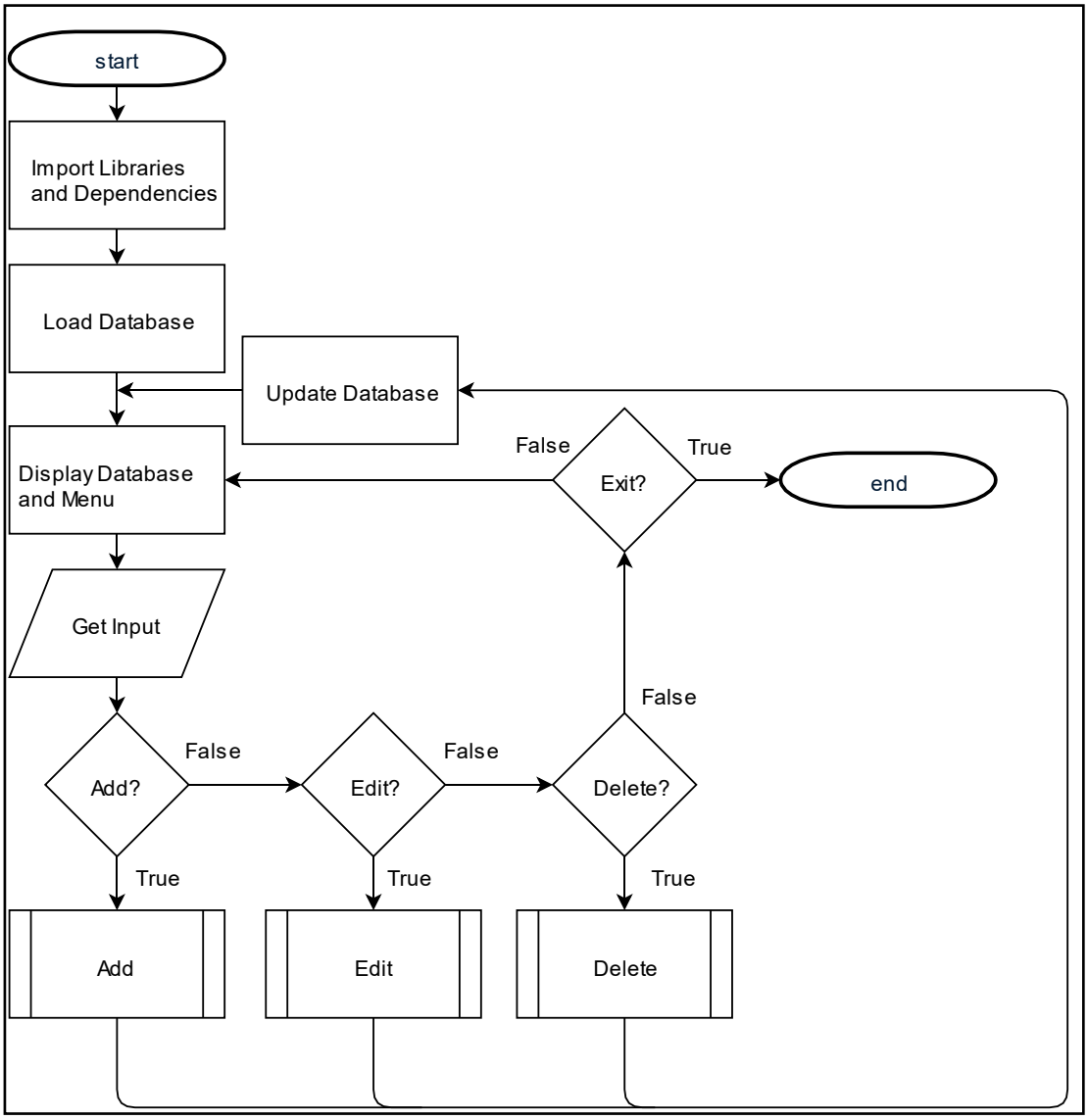
Selama program tidak di close, program akan terus mengecek folder “./data/request” dan bila ada file berekstensi .facedat didalamnya. Program akan membaca file itu, langsung menghapusnya dan mengecek dengan *local database* yang terletak di “./data/images”, bila kecocokannya lebih dari 60% (selisih nya kurang dari 40%), maka program akan memanggil Python untuk melakukan *logging* dan *feedback* ke client.



Gambar 3.4 Flowchart database.py

Flowchart diatas menggambarkan alur *script Python* database.py yang dipanggil oleh C++. Hal yang dilakukan di *script* ini adalah mengambil *parameter* yang dilempar (dalam kasus ini dari program C++) dan selanjutnya mengeksekusi perintah PostGre. Yang pertama membuka koneksi ke database.db, selanjutnya

dijalankan perintah SELECT, bila hasilnya kosong maka dilakukan INSERT ke *database*, bila tidak kosong maka dilakukan UPDATE, setelah itu di commit.

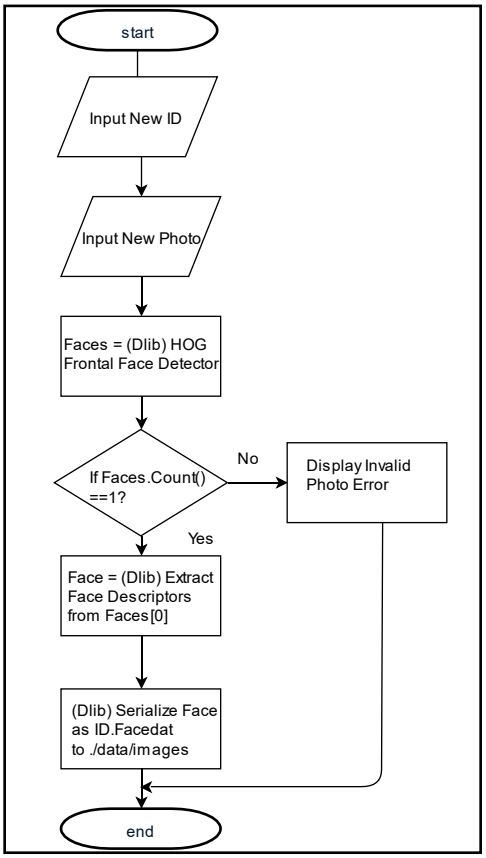


Gambar 3.5 Flowchart Program Server Database

Dari flowchart diatas, mengambarkan alur kerja program server database yang terletak di kantor pusat. “Import Libraries and Dependencies” adalah melakukan

import terhadap library OpenCV, Dlib, dan Dirent beserta file-file yang dibutuhkan untuk *runtime*.

Program saat baru dijalankan akan membuat *runtime database*, yang berfungsi sebagai wadah data sementara. Program akan terus berjalan sampai user memilih menu exit, setiap pilihan pada menu akan memanggil masing-masing proses lain. Setiap proses selesai, *runtime database* program ini akan di *update*.

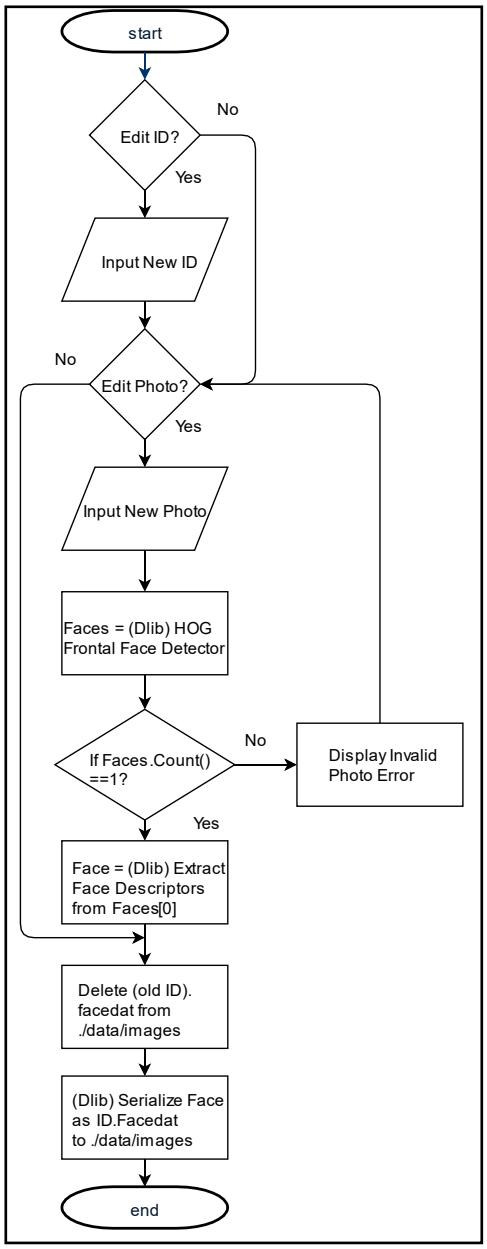


Gambar 3.6 Flowchart Method Add

Proses ini akan meminta id sebagai nama file dan meminta foto untuk dimasukkan ke *database*, bila foto berada di direktori yang sama dengan program

database, maka cukup menginput nama file, tapi bila foto tersebut berada di luar direktori, maka harus menginput alamatnya. Foto tersebut kemudian di cek jumlah wajahnya memakai Dlib HOG face detector.

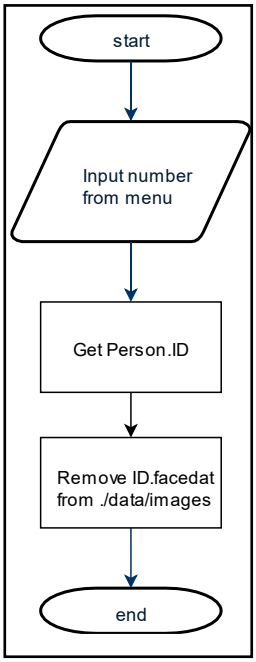
Bila foto memiliki kurang atau lebih dari satu wajah, maka program akan mengeluarkan *invalid photo error* dan langsung kembali ke menu awal. Bila foto memiliki tepat satu wajah, maka program akan menggunakan resnet *neural network* untuk mengambil 128 pengukuran wajah. Lalu program akan melakukan serialization terhadap data tersebut kedalam suatu file yang namanya sesuai id yang diinput dan extensinya adalah ".facedat", file tersebut akan ditaruh di dalam "./data/images" sebagai *local database*.



Gambar 3.7 Flowchart Method Edit

Proses ini akan melakukan hal yang sama dengan proses add, perbedaannya ada beberapa, yang pertama adalah program menanyakan apakah yang perlu diganti dengan menjawab Y/N (yes or no). Jika user mengganti id, maka id yang lama akan disimpan sementara. Jika user mengganti foto, program akan meminta foto dengan cara yang sama. Perbedaannya adalah bila fotonya invalid, maka program akan menanyakan kembali apakah masih ingin *edit* foto atau tidak. Bila fotonya ter-*edit*, data yang ada di *runtime database* akan di *update*.

File yang menggunakan id yang lama, walau tidak di *edit*, akan di hapus dan program akan melakukan *serialize* ulang dari *runtime database.* Nama file tersebut akan sesuai dengan id yang di input ataupun bila tidak mengganti id akan sesuai dengan id lama, dan isi dari file tersebut akan sesuai dengan *runtime database.*

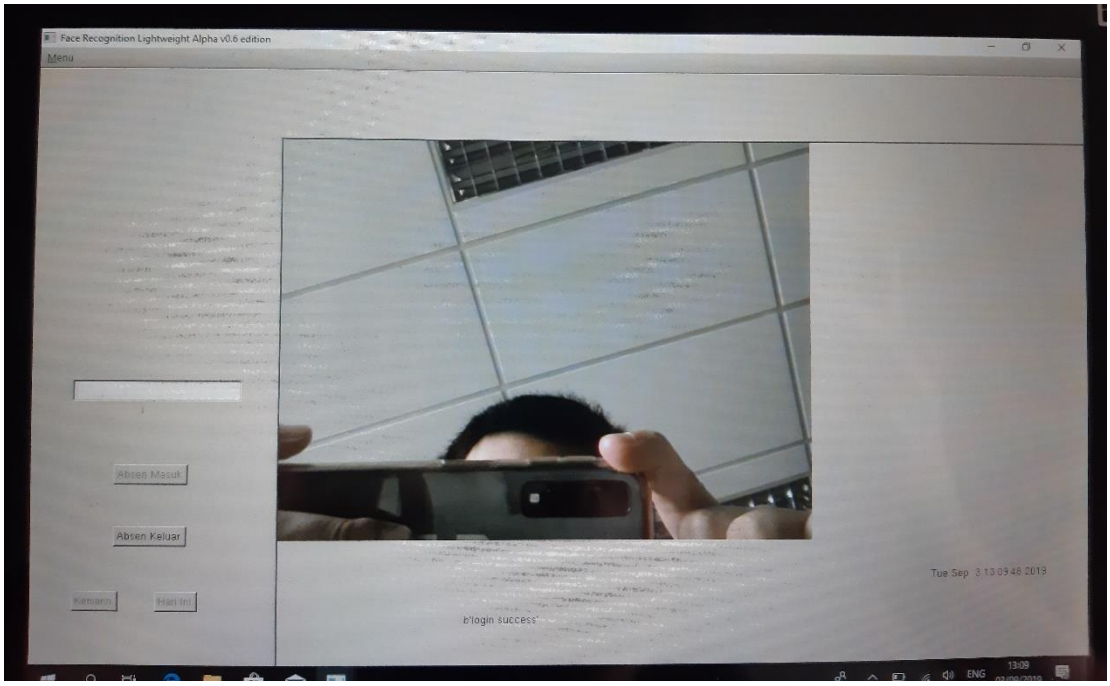


Gambar 3.8 Flowchart Method Delete

Proses ini akan menggunakan menu sebelumnya (*main* menu) yang menampilkan seluruh data yang ada di *database*. Karena datanya berurutan dan mempunyai nomor, *user* tinggal memilih satu dari data-data tersebut, setelah dapat maka program akan mendapatkan id dari pilihan itu dan langsung menghapus file yang bernama id yang berekstensi facedat di dalam "./data/images".

B. Implementasi

Berikut merupakan print screen antarmuka *prototype* aplikasi *face recognition* yang merupakan hasil implementasi yang dikerjakan selama periode magang. Ditengah berisi webcam live video dengan kotak merah deteksi wajah, dibawahnya berisi feedback dari server bila terkoneksi dengan *server*. Di pojok kiri atas berisi template menu yang belum di implementasi selain “about” dan di pojok kanan bawah adalah tanggal dan jam yang di *update* secara *realtime*.



Gambar 3.9 Rancang Antarmuka

```
template <template <int, template<typename>class, int, typename> class block, int N, template<typename>class BN, typename SUBNET>
using residual = add_prev1<block<N, BN, 1, tag1<SUBNET>>>;

template <template <int, template<typename>class, int, typename> class block, int N, template<typename>class BN, typename SUBNET>
using residual_down = add_prev2<avg_pool<2, 2, 2, 2, skip1<tag2<block<N, BN, 2, tag1<SUBNET>>>>>>;

template <int N, template <typename> class BN, int stride, typename SUBNET>
using block  = BN<con<N, 3, 3, 1, 1, relu<BN<con<N, 3, 3, stride, stride, SUBNET>>>>>;

template <int N, typename SUBNET> using ares      = relu<residual<block, N, affine, SUBNET>>;
template <int N, typename SUBNET> using ares_down = relu<residual_down<block, N, affine, SUBNET>>;

template <typename SUBNET> using alevel0 = ares_down<256, SUBNET>;
template <typename SUBNET> using alevel1 = ares<256, ares<256, ares_down<256, SUBNET>>>;
template <typename SUBNET> using alevel2 = ares<128, ares<128, ares_down<128, SUBNET>>>;
template <typename SUBNET> using alevel3 = ares<64, ares<64, ares<64, ares_down<64, SUBNET>>>>;
template <typename SUBNET> using alevel4 = ares<32, ares<32, ares<32, SUBNET>>>;

using anet_type = loss_metric<fc_no_bias<128, avg_pool_everything<
    alevel0<
    alevel1<
    alevel2<
    alevel3<
    alevel4<
    max_pool<3, 3, 2, 2, relu<affine<con<32, 7, 7, 2, 2,
    input_rgb_image_sized<150>
    >>>>>>>>>>>;

// Network Definition
//////////////////////////////////////////////////////////////////////////////////////////////
template <long num_filters, typename SUBNET> using con5d = con<num_filters, 5, 5, 2, 2, SUBNET>;
template <long num_filters, typename SUBNET> using con5  = con<num_filters, 5, 5, 1, 1, SUBNET>;

template <typename SUBNET> using downsampler  = relu<affine<con5d<32, relu<affine<con5d<32, relu<affine<con5d<16, SUBNET>>>>>>>>>;
template <typename SUBNET> using rcon5  = relu<affine<con5<45, SUBNET>>>;

//using net_type = loss_mmod<con<1, 9, 9, 1, 1, rcon5<rcon5<rcon5<downsampler<input_rgb_image_pyramid<pyramid_down<6>>>>>>>;
//////////////////////////////////////////////////////////////////////////////////////////////
```

Gambar 3.10 Source Code Neural Network Declaration

Gambar diatas merupakan foto *source code* dari deklarasi *neural network* yang dipakai di aplikasi, mengikuti *template* yang disediakan Dlib.

```
std::string exec(const char* cmd) {
    std::string result = "";
    try {
        std::array<char, 128> buffer;
        std::unique_ptr<FILE, decltype(&_pclose)> pipe(_popen(cmd, "r"), _pclose);
        if (!pipe) {
            throw std::runtime_error("exec() failed!");
        }
        while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr) {
            result += buffer.data();
        }
    }
    catch(exception e){
        result = e.what();
    }
    return result;
}
```

Gambar 3.11 Source Code Method Exec

Gambar diatas merupakan foto *source code* metode exec yang dipakai untuk memanggil semua kode Python kecuali pembuatan *service* FLASK API dan "initialize database" yang ada di server. Menggunakan *try* & *catch* untuk menampilkan *error* dan mencegah program keluar dari *loop*.

```
char* StringToCharArray(string x) {
    char cstr[100];
    strcpy(cstr, x.c_str());
    return cstr;
}
```

Gambar 3.12 Source Code Method stringToCharArray

Gambar diatas merupakan foto *source code* metode stringToCharArray yang dipakai untuk mengkonversi string menjadi *char array*.

```
struct people {
    string id;
    matrix<float, 0, 1> face;
    people() {
        id = ""; //default constructor, used to generate empty people
    }
    people(string id, matrix<float, 0, 1> muka) {
        id = id; //specialized constructor, used to make new people with datas
        face = muka;
    }
};
```

Gambar 3.13 Source Code Struct people

Gambar diatas merupakan foto *source code* dari *struct* people yang dapat menampung id user, dan *face descriptor* matrix-128 yang berisi angka float 0-1. Matrix inilah yang akan dibandingkan untuk mengenali wajah yang tidak dikenal.

```
void faceCompareProcess(std::vector<matrix<rgb_pixel>> faces,Win *win) {
    // Detect faces
    // This call asks the DNN to convert each face image in faces into a 128D vector.
    // In this 128D vector space, images from the same person will be close to each other
    // but vectors from different people will be far apart. So we can use these vectors to
    // identify if a pair of images are from the same person or from different people.
    std::vector<matrix<float, 0, 1>> face_descriptors;
    try {
        if (faces.size() > 0) {
            face_descriptors = net(faces);
            matrix<float, 0, 1> face;
            for (size_t i = 0; i < face_descriptors.size(); ++i)
            {
                string filename = std::to_string(requestNum) + ".facedat";
                try {
                    if (requestNum >= 2000000) {
                        requestNum = 0;
                    }
                    face = face_descriptors[i];
                    serialize("./data/clientrequest/" + filename) << face;
                    requestNum++;
                    //Python Upload
                    string param = "data/clientrequest/" + filename;//put param here
                    string syntax = "python logger.py \"" + iphost + "\" \"" + param + "\"";
                    string result = exec(StringToCharArray(syntax));
                    if (result != "") {
                        (*win).add_feedback(result);
                    }
                }
                catch (exception e) {
                    cerr << "Something error in the serialize of faceCompareProcess thread" << endl;
                    cout << e.what() << endl;
                }
            }
        }
    }
    catch (exception e) {
        cerr << "Something error in the faceCompareProcess thread" << endl;
        cout << e.what() << endl;
    }
}
```

Gambar 3.14 Source Code faceCompareProcess

Gambar tersebut merupakan foto *source code* dari *method* faceCompareProcess yang akan mengambil data yang dilempar oleh method detectFace beserta alamat *prototype window*. Setelah dapat data yang dilempar, data dikonversi oleh Dlib resnet dan di seriailize ke dalam file sebelum dikirim menggunakan Python. Hasil dari pengiriman dengan Python ini akan ditampilkan ke *prototype window* dengan cara mengakses alamat dan memasangnya kesana.

```
void detectFace(Win *win) {
    while (1) {
        try {
            (*win).clear_overlay();
            matrix<rgb_pixel> currentFrame = dlibMatrix;
            std::vector<dlib::rectangle> facePool = detector(currentFrame); //hog
            //std::vector<dlib::mmod_rect>facePool = mmodFaceDetector(dlibMatrix); //mmod
            if (facePool.size() > 0) {
                std::vector<matrix<rgb_pixel>> faces;
                for (dlib::rectangle face : facePool)
                {
                    matrix<rgb_pixel> face_chip = matrix<rgb_pixel>();
                    dlib::full_object_detection shape = sp(currentFrame, face);
                    (*win).add_overlay(face);
                    extract_image_chip(currentFrame, get_face_chip_details(shape, 150, 0.25), face_chip);
                    faces.push_back(move(face_chip));
                }
                std::thread t = thread(faceCompareProcess, faces);
                t.detach();
            }
        }
        catch (exception e) {
            cerr << "Something error in the thread detectFace" << endl;
            cout << e.what() << endl;
        }
        Sleep(1000);
    }
}
```

Gambar 3.15 Source Code detectFace

Gambar diatas merupakan foto *source code* dari *method* detectFace yang akan mengambil satu gambar/frame yang akan di cek lokasi mukanya menggunakan HOG face detector (MMOD untuk NVIDIA CUDA *build*). Untuk setiap muka yang di deteksi, program akan menampilkan satu kotak merah deteksi muka dan membuat satu *thread* perbandingan wajah faceCompareProcess untuk dibandingkan dengan database di server.

```
int main()
{
    ifstream inFile;
    inFile.open("iphost.txt");
    if (!inFile) {
        cout << "Unable to open iphost.txt" << endl;
    }
    inFile >> iphost;
    inFile.close();
    unsigned int option=0;
    detector = get_frontal_face_detector(); //hog
    //deserialize("mmod_human_face_detector.dat") >> mmodFaceDetector;
    deserialize("shape_predictor_5_face_landmarks.dat") >> sp;
    deserialize("dlib_face_recognition_resnet_model_v1.dat") >> net;
    people temp;
    mkdir("./data");
    mkdir("./data/clientrequest");
    system("CLS"); //clearscreen
    Win my_window;
    // Grab and process frames until the main window is closed by the user.
    // Run the face detector on the image of our action heroes, and for each face extract a
    // copy that has been normalized to 150x150 pixels in size and appropriately rotated
    // and centered.
    std::thread t = thread(detectFace, &my_window);
    t.detach();
    cv::VideoCapture cap(1);
    if (!cap.isOpened())
    {
        cap = cv::VideoCapture(0);
    }
```

Gambar 3.16 Source Code Client Initializer

Gambar diatas merupakan foto *source code* dari *initializer* pada *client*, disini semua *dependencies* dimuat terlebih dahulu, seperti iphost.txt, HOG face detector, shape predictor 5 landmark Dlib dan Dlib face recognition resnet. Disini juga melakukan *generate* terhadap folder yang dibutuhkan pada saat *runtime*. Sebelum masuk ke *main loop*, program akan membuka *prototype window*, membuat *thread* tambahan untuk pendeteksi muka detectFace dan membuka saluran videocapture opencv yang mengambil akses ke *webcam*.

```
while (!my_window.is_closed()) {
    if (!cap.isOpened())
    {
        cerr << "Unable to connect to camera" << endl;
        return -1;
    }
    try {
        // Grab a frame
        cv::Mat temp;
        if (!cap.read(temp))
        {
            break;
        }
        // Turn OpenCV's Mat into something dlib can deal with.  Note that this just
        // wraps the Mat object, it doesn't copy anything.  So cimg is only valid as
        // long as temp is valid.  Also don't do anything to temp that would cause it
        // to reallocate the memory which stores the image as that will make cimg
        // contain dangling pointers.  This basically means you shouldn't modify temp
        // while using cimg.
        time(&rawtime);
        timeinfo = localtime(&rawtime);
        static char buffer[80];
        strftime(buffer, 80, "%c", timeinfo);
        my_window.set_time(buffer);
        cv_image<bgr_pixel> cimg(temp);
        assign_image(dlibMatrix, cimg);
        my_window.set_image(cimg);
    }
    catch (exception e) {
        cerr << "Something error in the frame" << endl;
        cout << e.what() << endl;
    }
}
return 0;
```

Gambar 3.17 Source Code Client

Gambar diatas merupakan *main loop* dari program *client*, sekaligus lanjutan dari *initializer* pada *client*. Jika akses *webcam* hilang, maka program akan melakukan return -1 dan berhenti. Selama akses webcam masih ada, program akan terus menerus mengambil frame dari webcam dan menampilkannya ke client. Waktu juga akan terus di update dan ditampilkan di *prototype window*. Dan setiap kali iterasi, gambar

framenya akan disimpan di *global variable* yang akan diakses oleh thread DetectFace.

```
void loadDatabase() {
    try {
        people temp;
        //load all dat here, per person and add them to the person vector
        DIR* pDIR;
        struct dirent* entry;
        person.clear();
        if (pDIR = opendir("./data/images")) {
            while (entry = readdir(pDIR)) {
                string filename = entry->d_name;
                if (strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0 && filename.find(".facedat") != std::string::npos) {
                    deserialize("./data/images/"+filename) >> temp.face; //load from file
                    std::size_t pos = filename.find(".facedat");
                    temp.id = filename.substr(0, pos);
                    person.push_back(temp);
                }
            }
            closedir(pDIR);
        }
    }
    catch (serialization_error & e)
    {
        cerr << "One or more files seems doesn't have face, check integrity of the files" << endl;
        cout << endl << e.what() << endl;
    }
}
```

Gambar 3.18 Source Code LoadDatabase

Gambar diatas merupakan foto dari *source code* LoadDatabase, dimana *server* akan melakukan *loading* terhadap *database runtime* dengan cara membaca setiap file di "./data/images/" dan distrukturisasi ke dalam *struct* people.

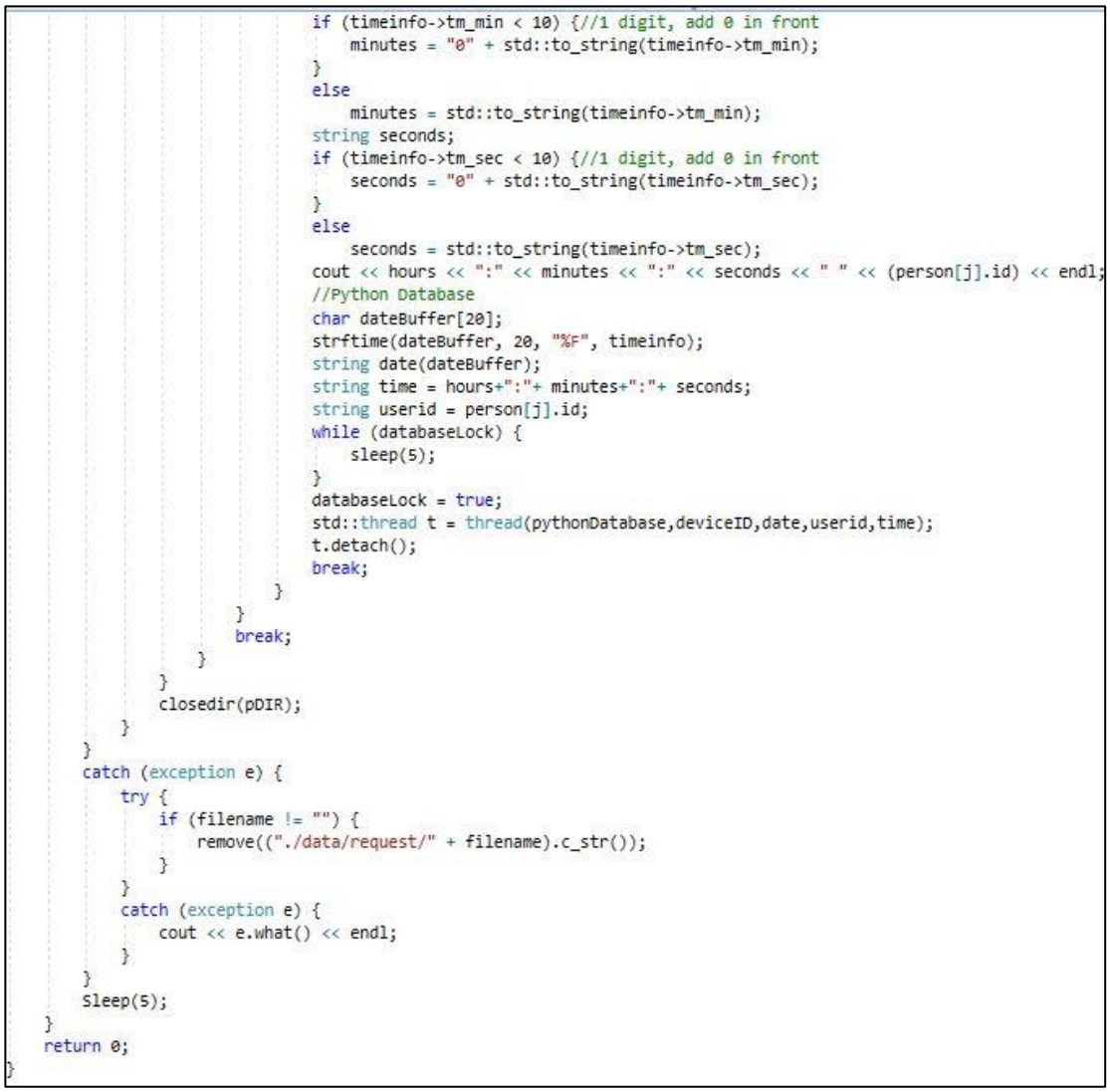
```
//Python database initializer
std::thread t1 = thread(pythonInitializeDatabase);
t1.join();
//Python Service Boot Up
std::thread t = thread(pythonServiceBootUp);
t.detach();
deserialize("dlib_face_recognition_resnet_model_v1.dat") >> net;
person.resize(0);
mkdir("./data");
mkdir("./data/images");
mkdir("./data/request");
loadDatabase();
int hourRan = (std::rand() % 4);
char buffer[80];
time(&rawtime);
timeinfo = localtime(&rawtime);
strftime(buffer, 80, "%A %d %B %Y", timeinfo);
puts(buffer);
```

Gambar 3.19 Source Code Server Initializer

Gambar diatas merupakan foto dari source code Server Initializer, dimana program C++ ini akan memanggil dua *thread* yang akan menjalankan Python code dan melakukan *import dependencies* ke aplikasi yang kali ini hanyalah Dlib resnet. Di tahap ini juga program akan men-*generate* folder yang dibutuhkan untuk runtime, dan tanggal untuk hari ini, serta menentukan waktu untuk melakukan refresh di main loop.

```
while (1)
{
    if (updated == false && timeinfo->tm_hour == hourRan) {
        loadDatabase();
    }
    else if (updated == true && timeinfo->tm_hour > 22) {
        updated = false;
    }
    if (timeinfo->tm_hour == 0) {
        strftime(buffer, 80, "%A %d %B %Y", timeinfo);
        puts(buffer);
    }
    string filename;
    try {
        time(&rawtime);
        timeinfo = localtime(&rawtime);
        filename = "";
        matrix<float, 0, 1> face;
        //load all dat here, per person and add them to the person vector
        DIR* pDIR;
        struct dirent* entry;
        if (pDIR = opendir("./data/request")) {
            while (entry = readdir(pDIR)) {
                filename = entry->d_name;
                if (strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0 && filename.find(".facedat") != std::string::npos) {
                    deserialize("./data/request/" + filename) >> face; //load from file
                    std::size_t pos = filename.find(".facedat");
                    string x = filename.substr(0, pos);
                    pos = x.find("_");
                    string deviceID = x.substr(0, pos);
                    //Delete the facedat file
                    remove(("./data/request/" + filename).c_str());
                    for (size_t j = 0; j < person.size(); ++j)
                    {
                        if (length(face - person[j].face) < 0.4) // 0.4 because 0.6 standard does false positive
                        {
                            string hours;
                            if (timeinfo->tm_hour < 10) {//1 digit, add 0 in front
                                hours = "0" + std::to_string(timeinfo->tm_hour);
                            }
                            else
                                hours = std::to_string(timeinfo->tm_hour);
                            string minutes;
```

Gambar 3.20 Source Code Server (1)



```
                        if (timeinfo->tm_min < 10) {//1 digit, add 0 in front
                            minutes = "0" + std::to_string(timeinfo->tm_min);
                        }
                        else
                            minutes = std::to_string(timeinfo->tm_min);
                        string seconds;
                        if (timeinfo->tm_sec < 10) {//1 digit, add 0 in front
                            seconds = "0" + std::to_string(timeinfo->tm_sec);
                        }
                        else
                            seconds = std::to_string(timeinfo->tm_sec);
                        cout << hours << ":" << minutes << ":" << seconds << " " << (person[j].id) << endl;
                        //Python Database
                        char dateBuffer[20];
                        strftime(dateBuffer, 20, "%F", timeinfo);
                        string date(dateBuffer);
                        string time = hours+":"+ minutes+":"+ seconds;
                        string userid = person[j].id;
                        while (databaseLock) {
                            sleep(5);
                        }
                        databaseLock = true;
                        std::thread t = thread(pythonDatabase,deviceID,date,userid,time);
                        t.detach();
                        break;
                    }
                }
                break;
            }
        }
        closedir(pDIR);
    }
}
catch (exception e) {
    try {
        if (filename != "") {
            remove(("./data/request/" + filename).c_str());
        }
    }
    catch (exception e) {
        cout << e.what() << endl;
    }
}
Sleep(5);
}
return 0;
}
```

Gambar 3.21 Source Code Server (2)

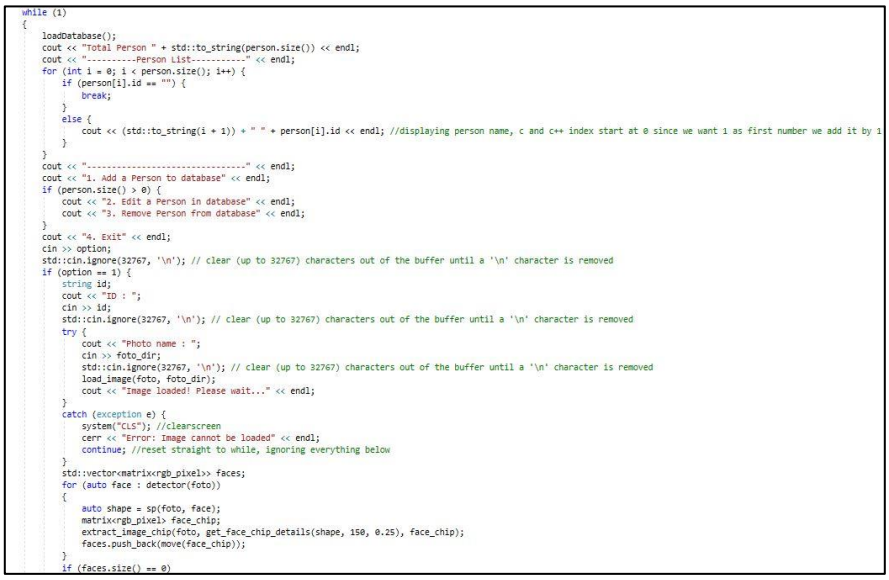
Gambar-gambar diatas merupakan foto *source code* untuk *main loop server*. Setiap pergantian hari program akan mencetak tanggal hari itu dan setiap hari dengan waktu yang sudah di initialize (jam 0-4) pagi maka program akan melakukan *loading* ulang terhadap *database* runtime. Untuk setiap file ".facedat" yang berada dalam ".data/request", server akan membaca device id dan request id dari filename,

membaca isinya dan langsung mendelete file itu, Setelah itu, server membandingkan isi filenya dengan *database runtime*, bila perbedaan dari kedua file lebih kecil dari 40% (kemiripan 60%+) maka dicetak ke *server console* dan membuka *script* Python untuk menulis ke *database*. Setiap pembacaan file memiliki *delay* 5 *miliseconds* (*sleep*) agar *cpu* tidak *occupied* terus menerus.

```
int option;
frontal_face_detector detector = get_frontal_face_detector();
shape_predictor sp;
deserialize("shape_predictor_5_face_landmarks.dat") >> sp;
anet_type net;
deserialize("dlib_face_recognition_resnet_model_v1.dat") >> net;
person.resize(0);
people temp;
mkdir("./data");
mkdir("./data/images");
string foto_dir;
matrix<rgb_pixel> foto; //contain photo in a rgb matrix
system("CLS"); //clearscreen
```

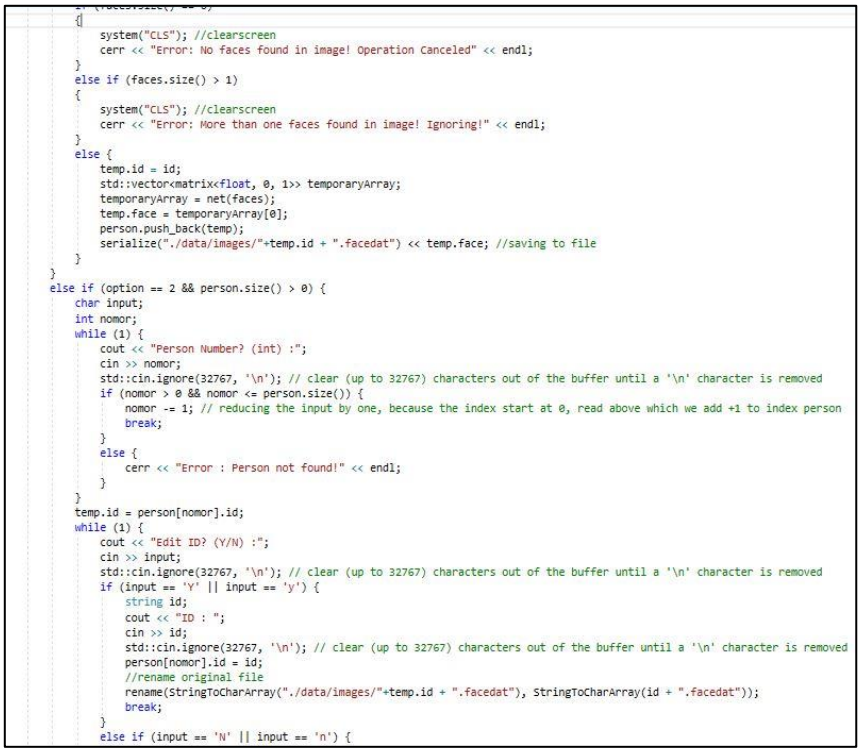
Gambar 3.22 Source Code Server Database Initializer

Gambar diatas merupakan foto source code untuk *server database initializer*, di tahap ini program melakukan *import dependencies* yaitu shape predictor 5 face landmark dan dlib resnet. Setelah melakukan import, program akan meng-*generate folder-folder* yang dibutuhkan saat *runtime* dan melakukan *memory space reserve* untuk menyimpan foto yang akan diinput nantinya



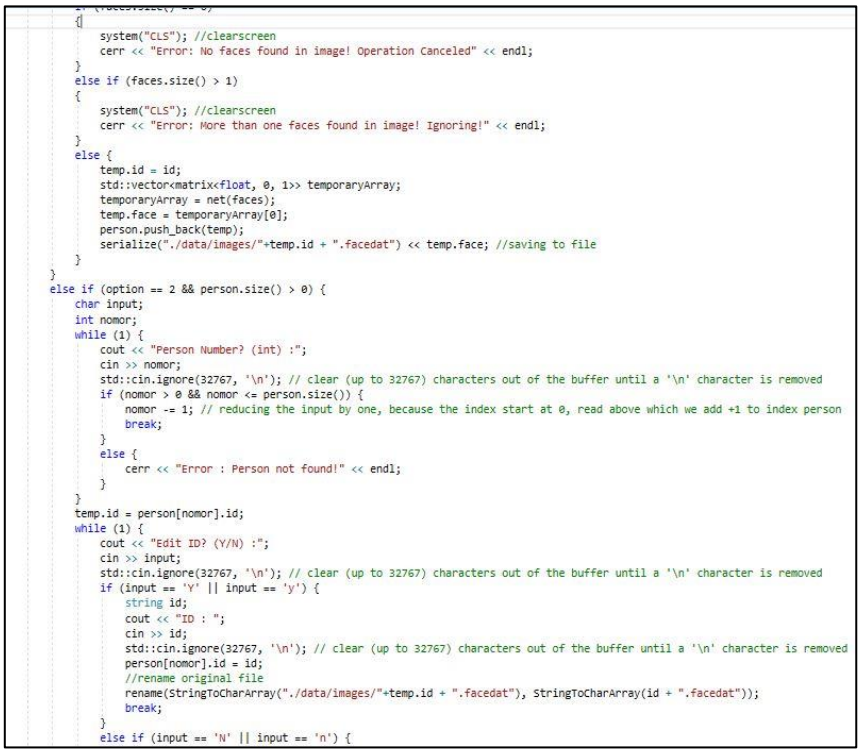
```
while (1)
{
    loadDatabase();
    cout << "Total Person " + std::to_string(person.size()) << endl;
    cout << "---------Person List----------" << endl;
    for (int i = 0; i < person.size(); i++) {
        if (person[i].id == "") {
            break;
        }
        else {
            cout << (std::to_string(i + 1)) + " " + person[i].id << endl; //displaying person name, c and c++ index start at 0 since we want 1 as first number we add it by 1
        }
    }
    cout << "-------------------------------" << endl;
    cout << "1. Add a Person to database" << endl;
    if (person.size() > 0) {
        cout << "2. Edit a Person in database" << endl;
        cout << "3. Remove Person from database" << endl;
    }
    cout << "4. Exit" << endl;
    cin >> option;
    std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
    if (option == 1) {
        string id;
        cout << "ID : ";
        cin >> id;
        std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
        try {
            cout << "Photo name : ";
            cin >> foto_dir;
            std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
            load_image(foto, foto_dir);
            cout << "Image loaded! Please wait..." << endl;
        }
        catch (exception e) {
            system("CLS"); //clearscreen
            cerr << "Error: Image cannot be loaded" << endl;
            continue; //reset straight to while, ignoring everything below
        }
        std::vector<matrix<rgb_pixel>> faces;
        for (auto face : detector(foto))
        {
            auto shape = sp(foto, face);
            matrix<rgb_pixel> face_chip;
            extract_image_chip(foto, get_face_chip_details(shape, 150, 0.25), face_chip);
            faces.push_back(move(face_chip));
        }
        if (faces.size() == 0)
```

Gambar 3.23 Source Code Server Database (1)



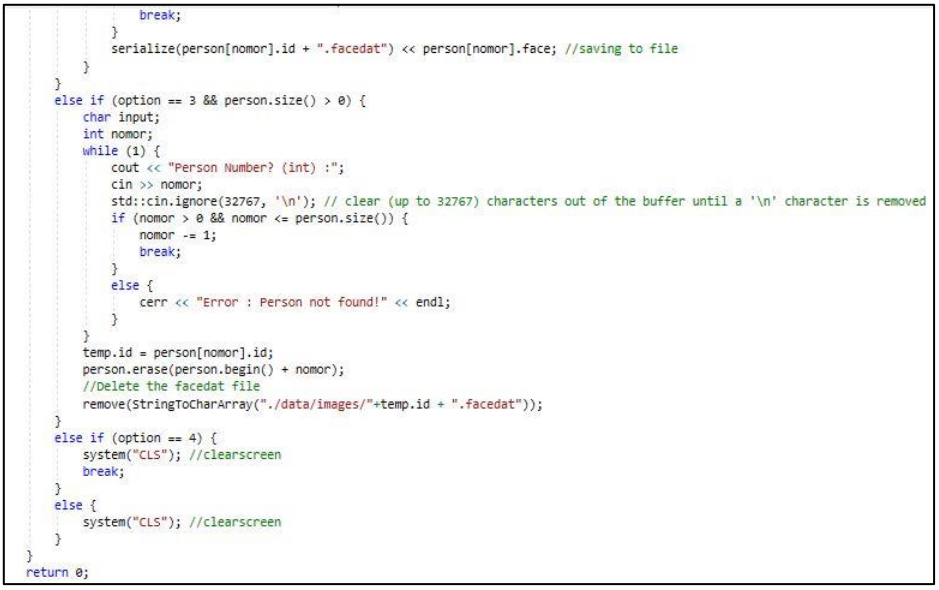
```
        {
            system("CLS"); //clearscreen
            cerr << "Error: No faces found in image! Operation Canceled" << endl;
        }
        else if (faces.size() > 1)
        {
            system("CLS"); //clearscreen
            cerr << "Error: More than one faces found in image! Ignoring!" << endl;
        }
        else {
            temp.id = id;
            std::vector<matrix<float, 0, 1>> temporaryArray;
            temporaryArray = net(faces);
            temp.face = temporaryArray[0];
            person.push_back(temp);
            serialize("./data/images/"+temp.id + ".facedat") << temp.face; //saving to file
        }
    }
    else if (option == 2 && person.size() > 0) {
        char input;
        int nomor;
        while (1) {
            cout << "Person Number? (int) :";
            cin >> nomor;
            std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
            if (nomor > 0 && nomor <= person.size()) {
                nomor -= 1; // reducing the input by one, because the index start at 0, read above which we add +1 to index person
                break;
            }
            else {
                cerr << "Error : Person not found!" << endl;
            }
        }
        temp.id = person[nomor].id;
        while (1) {
            cout << "Edit ID? (Y/N) :";
            cin >> input;
            std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
            if (input == 'Y' || input == 'y') {
                string id;
                cout << "ID : ";
                cin >> id;
                std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
                person[nomor].id = id;
                //rename original file
                rename(StringToCharArray("./data/images/"+temp.id + ".facedat"), StringToCharArray(id + ".facedat"));
                break;
            }
            else if (input == 'N' || input == 'n') {
```

Gambar 3.24 Source Code Server Database (2)



```
            {
                system("CLS"); //clearscreen
                cerr << "Error: No faces found in image! Operation Canceled" << endl;
            }
            else if (faces.size() > 1)
            {
                system("CLS"); //clearscreen
                cerr << "Error: More than one faces found in image! Ignoring!" << endl;
            }
            else {
                temp.id = id;
                std::vector<matrix<float, 0, 1>> temporaryArray;
                temporaryArray = net(faces);
                temp.face = temporaryArray[0];
                person.push_back(temp);
                serialize("./data/images/"+temp.id + ".facedat") << temp.face; //saving to file
            }
        }
        else if (option == 2 && person.size() > 0) {
            char input;
            int nomor;
            while (1) {
                cout << "Person Number? (int) :";
                cin >> nomor;
                std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
                if (nomor > 0 && nomor <= person.size()) {
                    nomor -= 1; // reducing the input by one, because the index start at 0, read above which we add +1 to index person
                    break;
                }
                else {
                    cerr << "Error : Person not found!" << endl;
                }
            }
            temp.id = person[nomor].id;
            while (1) {
                cout << "Edit ID? (Y/N) :";
                cin >> input;
                std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
                if (input == 'Y' || input == 'y') {
                    string id;
                    cout << "ID : ";
                    cin >> id;
                    std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
                    person[nomor].id = id;
                    //rename original file
                    rename(StringToCharArray("./data/images/"+temp.id + ".facedat"), StringToCharArray(id + ".facedat"));
                    break;
                }
                else if (input == 'N' || input == 'n') {
```

Gambar 3.25 Source Code Server Database (3)



```
                    break;
                }
                serialize(person[nomor].id + ".facedat") << person[nomor].face; //saving to file
            }
        }
        else if (option == 3 && person.size() > 0) {
            char input;
            int nomor;
            while (1) {
                cout << "Person Number? (int) :";
                cin >> nomor;
                std::cin.ignore(32767, '\n'); // clear (up to 32767) characters out of the buffer until a '\n' character is removed
                if (nomor > 0 && nomor <= person.size()) {
                    nomor -= 1;
                    break;
                }
                else {
                    cerr << "Error : Person not found!" << endl;
                }
            }
            temp.id = person[nomor].id;
            person.erase(person.begin() + nomor);
            //Delete the facedat file
            remove(StringToCharArray("./data/images/"+temp.id + ".facedat"));
        }
        else if (option == 4) {
            system("CLS"); //clearscreen
            break;
        }
        else {
            system("CLS"); //clearscreen
        }
    }
    return 0;
```

Gambar 3.26 Source Code Server Database (4)

Gambar-gambar diatas merupakan *source code* program *server database*, dimana program akan melakukan *listing* terhadap semua data yang sudah ada di folder "./data/images" dan program dapat menambah, mengedit atau membuang data yang ada di folder tersebut dengan menu menu yang disediakan.

```
import sys
import psycopg2
from psycopg2 import Error

deviceID = str(sys.argv[1])
tanggal = str(sys.argv[2])
ttime = str(sys.argv[3])
id = str(sys.argv[4])
jam = str(sys.argv[5])
flag = int(sys.argv[6])

def create_record():
    ttime2=ttime.replace("@"," ")
    ttime3=ttime2.replace("-",":")
    transactionTime=ttime3.replace(":","-",2)
    sql = ''' INSERT INTO absensi.absensi
    (device_id,aa_tanggal,aa_tgl_transaksi,aa_nik,aa_jam,aa_status) VALUES('{}','{}','{}','{}','{}','{}');'''
    .format(deviceID,tanggal,transactionTime,id,jam,flag)
    conn = psycopg2.connect(user = "postgres",
        password="admin",
        host="127.0.0.1",
        port="5432",
        database="absensi")
    cur = conn.cursor()
    cur.execute(sql)
    x=cur.lastrowid
    conn.commit()
    conn.close()
    return x

def main():
    # create a database connection
    create_record()

if __name__ == '__main__':
    main()
```

Gambar 3.27 Source Code database.py

Gambar diatas adalah foto *source code* initializedatabase.py, dimana *script* ini membuat koneksi ke database.db lalu mengeksekusi perintah PostGRE SELECT, bila kosong akan dilakukan perintah INSERT, bila tidak kosong akan dilakukan perintah UPDATE. Setelah selesai, *database* di *commit*.

```
import sys
import os
from urllib import parse, request
import requests
import base64
import json
import time
from datetime import datetime,date

empid=""
date=""
transactiontime=""
status=""

def function(p_url,p_param):
    details=""
    message=""
    try:
        date=datetime.today().strftime('%Y-%m-%d')
        req = requests.post(p_url,data=encoded,timeout=3)
        if req.status_code==200:
            message =req.content
            os.remove(p_param) #delete file
        else:
            req.raise_for_status()
    except requests.exceptions.ConnectionError as err:
        message = "Request timed out"
        details=message
    except requests.exceptions.HTTPError as err:
        if err.response.status_code==404:
            message = "not found"
            details=message
        elif err.response.status_code==500:
            message = "internal server error"
            details=message
    data={
        "NIK":empid,
        "Tanggal Absen":date,
        "Waktu Transaksi":transactiontime,
        "Status":status,
        "Keterangan":details
    }
    fout=open("log.txt",'a+')
    with fout as file:
        json.dump(data,fout,ensure_ascii=False)
    fout.close
    print(message)
```

Gambar 3.28 Source Code logger (1).py

```
if __name__=='__main__':
    req=None
    url = str(sys.argv[1])
    param = sys.argv[2]
    ttime=os.path.getmtime(param)
    transactiontime=time.strftime('%Y-%m-%d@%H-%M-%S',time.localtime(ttime))
    if len(sys.argv) == 5: #masuk
        #masuk=sys.argv[3]
        empid=sys.argv[4]
        fin = open(param, 'rb')
        files = {'file': fin}
        data=fin.read()
        fin.close()
        status=1
        encoded = base64.b64encode(data)
        fin=open("deviceID.txt",'r')
        data=fin.readline()
        fin.close()
        url+="masuk/"+data+"/"+transactiontime+"/"+empid
        a=function(url,param)

    elif len(sys.argv)==6: #keluar
        #keluar=sys.argv[3]
        kemarin=sys.argv[4]
        empid=sys.argv[5]
        fin = open(param, 'rb')
        files = {'file': fin}
        data=fin.read()
        fin.close()
        status=2
        encoded = base64.b64encode(data)
        fin=open("deviceID.txt",'r')
        data=fin.readline()
        fin.close()
        url+="keluar/"+data+"/"+transactiontime+"/"+empid+"/"+kemarin
        a=function(url,param)
```

Gambar 3.29 Source Code logger (2).py

Gambar diatas adalah foto *source code* logger.py, *script* ini mengambil url dan nama file dari parameter, lalu membaca isi binary dari file yang mempunyai nama file tersebut. Url ditambah dengan parameter *page* nya dan dengan deviceID yang didapat dengan cara membuka deviceID.txt yang terletak di folder yang sama, deviceID ini

harus di isi secara manual di tiap *client*. Setelah itu, *script* akan melakukan *post request* ke *server* dengan url yang sudah ditambah, datanya adalah isi *binary* file yang telah di *encode* dengan base64 dan mempunyai batas *timeout* selama 3 detik. Bila hasil feedbacknya mempunyai *status code* 200, maka file yang telah dibuka tersebut akan dihapus dan isi feedbacknya akan di cetak. Setelah itu hasil *response* di masukkan kedalam *log file*.

```
from flask import Flask, jsonify, send_file, request
import base64
import os
import sys
import psycopg2
from psycopg2 import Error
import time
from datetime import datetime,date

def select_record(deviceID,userID):
    tanggal=datetime.today().strftime('%Y-%m-%d')
    sql ='''' SELECT aa_nik, aa_jam, aa_status FROM
    absensi.absensi WHERE aa_nik='{}' AND device_id='{}' AND aa_tanggal ='{}';'''
    .format(userID,deviceID,tanggal)
    conn = psycopg2.connect(user = "postgres",
                password="admin",
                host="127.0.0.1",
                port="5432",
                database="absensi")
    cur = conn.cursor()
    cur.execute(sql)
    rows =  cur.fetchall()
    cur.close()
    conn.close()
    i=0
    for row in rows:
        now=datetime.now()
        jam = datetime.strptime(row[1],'%H:%M:%S')

        if(now.hour==jam.hour and now.minute ==jam.minute and abs(now.second-jam.second)<=1):
            i=1
    if i>0:
        return "login success"
    else:
        return "login failed"
    return "error occured"

app = Flask(__name__)
@app.route('/index')
def home():
    r = open(status.txt)
    return jsonify()
```

Gambar 3.30 Source Code sevice.py (1)

```
@app.route('/masuk/<deviceID>/<transactionTime>/<userID>',methods=['POST','GET'])
def parse(deviceID,transactionTime,userID):
    fout=open("serverlog.txt",'a+')
    with fout as file:
        file.write("request in %s %s\n" % (deviceID,datetime.now()))
    encoded = request.data
    path = '.\\data\\request'
    i=0
    for x in os.listdir(path):
        z=x.split('-')[3]
        y=z.split('.')[0]
        if(x.split('.')[0]=='facedat'):
            y=int(y)
            if y>=i:
                i=y+1
    if i>10000:
        i=0
    s=str(i)
    while len(s)<7:
        s="0"+s
    data=base64.b64decode(encoded)
    fout=open('data/request/'+'1'+'-'+deviceID+'-'+transactionTime+'_'+userID+'-'+s+'.facedat', 'wb')
    with fout as file:
        file.write(data)
    time.sleep(1)
    x=select_record(deviceID,userID)
    fout=open("serverlog.txt",'a+')
    with fout as file:
        file.write("request out %s %s\n" % (deviceID,datetime.now()))
    return x
```

Gambar 3.31 Source Code service.py (2)

```
@app.route('/keluar/<deviceID>/<transactionTime>/<userID>/<kemarin>',methods=['POST','GET'])
def parse2(deviceID,transactionTime,userID,kemarin):
    fout=open("serverlog.txt",'a+')
    with fout as file:
        file.write("request in %s %s\n" % (deviceID,datetime.now()))
    encoded = request.data
    path = '.\\data\\request'
    i=0
    for x in os.listdir(path):
        z=x.split('-')[3]
        y=z.split('.')[0]
        if(x.split('.')[0]=='facedat'):
            y=int(y)
            if y>=i:
                i=y+1
    if i>10000:
        i=0
    s=str(i)
    while len(s)<7:
        s="0"+s
    data=base64.b64decode(encoded)
    fout=open('data/request/'+'2'+'_'+kemarin+'-'+deviceID+'-'+transactionTime+'_'+userID+'-'+s+'.facedat', 'wb')
    with fout as file:
        file.write(data)
    time.sleep(1)
    x=select_record(deviceID,userID)
    fout=open("serverlog.txt",'a+')
    with fout as file:
        file.write("request out %s %s\n" % (deviceID,datetime.now()))
    return x

app.run(host="0.0.0.0",port=9880)
```

Gambar 3.32 Source Code service.py (3)

Gambar-gambar diatas adalah foto *source code* service.py, *script* ini mengambil request client dan membaca deviceID dan data yang dilempar. Setelah membaca data yang dilempar, data tersebut di decode dengan base64, lalu ditulis ke file facedat dengan deviceID di filenamenya agar dapat dibaca oleh program C++. Setelah itu program menunggu 1 detik (*sleep*) agar memberikan jeda untuk C++ menyelesaikan datanya, setelah itu, membuka koneksi dan melakukan query ke database.db di server, hasil querynya di return ke *client*.

C. Hasil Pengujian

Selama program dibuat, dilakukan *Significance Testing* dan *Hardware Testing*. *Significance Testing* yang dilakukan dengan 68 sampel data, dan *Hardware Testing* dilakukan dengan perangkat *tablet* Intel® Atom™ yang tersedia. Hasil pengujian bahwa 94% dari semua *login attempt* yang dilakukan mempunyai status berhasil, tidak ada *false positive* dan 6% wajah tidak dideteksi (dikarenakan *bad lighting* atau kemiringan wajah), perangkat yang tersedia mempunyai *webcam* yang tidak terlalu bagus dalam *auto-focus* dan *lighting* sehingga akurasi program dapat ditingkatkan di kemudian hari dengan mengganti perangkat.

3.3.2. Kendala yang Ditemukan

Dalam pembuatan prototype aplikasi, ditemukannya beberapa kendala seperti:

1. *Project Requirements* yang bertambah secara bertahap

2. Keterbatasan ilmu dan pengetahuan yang dimiliki
3. Keterbatasan spesifikasi client yang hanya merupakan tablet Intel® Atom™

3.3.3. Solusi atas Kendala yang Ditemukan

Dari kendala yang dihadapi ketika menjalani kerja magang, terdapat langkah-langkah solusi penyelesaian masalah yang diambil seperti:

1. Berkonsultasi dengan Bapak Erick Alviyendra untuk menentukan *roadmap* dan *to-do list* dalam seminggu
2. Mencari sumber referensi dari internet
3. Beradaptasi dan belajar dari rekan kerja yang lain