BAB III

METODOLOGI DAN PERANCANGAN SISTEM

3.1. Metodologi Penelitian

Metode penelitian yang digunakan dalam Implementasi penelitian ini antara lain adalah sebagai berikut.

a. Telaah Literatur

Pada tahap telaah literatur, pembelajaran terhadap teori-teori dan analisa penelitian terdahulu dilakukan. Teori ataupun analisa didapatkan dari berbagai sumber seperti jurnal dan buku yang berkaitan dengan penelitian. Teori dan analisa yang digunakan adalah *voice chatbot* Jacob, *computer vision (artificial intelligence, image processing,* dan *pattern recognition*), pengenalan wajah, *convolutional neural network*, FaceNet, Inception-ResNet-v1, *Least Recently Used*, dan *F-Measure*.

b. Analisis dan Perancangan Vision

Pada tahap analisis dan perancangan, dilakukan analisis terhadap *chatbot* Jacob yang sudah dibangun dan proses-proses yang terlibat dalam penelitian ini dapat dilihat pada Subbab 3.2. Setelah analisis pada *chatbot* Jacob dilakukan, proses analisis teori yang akan digunakan dalam modul dan juga analisis kebutuhan untuk integrasi dilakukan. Selain itu, data masukan, keluaran, alat penunjang penelitian yang akan digunakan dalam penelitian ini juga dianalisis dalam tahap ini.

Setelah mengumpulkan teori dan analisa terhadap kebutuhan penelitian, proses dilanjutkan ke perancangan dasar modul yang akan dibuat. Fungsionalitas modul ditetapkan pada tahap ini sesuai dengan analisis kebutuhan pada Subbab 3.2.

Rancangan antarmuka Vision, sistem penyimpanan, dan alur kerja dasar aplikasi juga dilakukan pada tahap ini. Modul yang dibangun diberi nama Vision.

c. Implementasi dan Integrasi Vision

Dalam tahap ini, dilakukan proses implementasi dari rancangan yang telah dilakukan pada tahap sebelumnya dan dibahas pada Bab 4 Subbab 4.2 dan integrasi dibahas pada Bab 4 Subbab 4.3. Pembuatan modul disesuaikan dengan spesifikasi dan fungsionalitas yang dijabarkan pada Bab 3 Subbab 3.2. Komponen utama yang diperlukan seperti API akan dibangun pada tahap ini.

d. Uji Coba dan Evaluasi

Vision yang telah selesai diimplementasi dan diintegrasikan dengan Jacob diuji pada tahap ini untuk memastikan terpenuhinya fungsionalitas sesuai perancangan sebelumnya. Evaluasi dilakukan dengan mengukur F-score untuk pengenalan wajah admin atau super admin saat proses registrasi dan *login* untuk mencapai tujuan dari penelitian ini. Uji coba dan Evaluasi dijabarkan pada Bab 4 Subbab 4.4 dan 4.5.

e. Penulisan Laporan

Proses dan hasil akhir dari penelitian akan dituangkan dalam bentuk laporan sebagai bukti atas penelitian yang telah dilakukan.

3.2. Analisis Kebutuhan Vision

Hasil analisis dalam pembuatan Vision terbagi menjadi tiga, yaitu kebutuhan masukan dan keluaran, fungsionalitas dan jenis pengguna *chatbot* Jacob, dan integrasi dengan Jacob. Pertama, kebutuhan masukan dari Vision ini adalah.

1. Data diri pengguna baru untuk registrasi berupa nama dan jenis hak akses.
2. Gambar wajah untuk dikenali pada saat Vision bekerja.

3. Jumlah gambar wajah latihan yang beragam, yaitu 20, 50, dan 100.

Keluaran dari modul ini berupa hasil registrasi dan hasil pengenalan wajah untuk *login*.

Kedua, analisis berdasarkan fungsionalitas dan jenis pengguna *chatbot* Jacob. Hasil dari analisis *chatbot* Jacob adalah tersedianya chatbot dengan tiga jenis hak akses, yaitu admin, super admin dan *user*. Sebelumnya, Jacob hanya dapat mengenali pengguna admin atau super admin dengan cara memasukkan kata sandi. Sedangkan pengguna akan dikategorikan sebagai *user* apabila tidak melakukan proses *login*. Dengan adanya Vision, diharapkan dapat memberikan pilihan proses otentikasi dengan mengenali wajah pengguna. Untuk mengenali wajah pengguna, maka wajah pengguna harus didaftarkan terlebih dahulu. Pengguna yang telah didaftarkan dan telah melewati fase pelatihan akan dapat dikenali. Apabila jenis pengguna yang dikenali adalah admin atau super admin, maka admin atau super admin akan langsung memiliki hak terhadap sistem pengaturan *Jacob*. Apabila pengguna dikenali sebagai *user*, maka pengguna hanya menerima sapaan. Dengan dapat dikenalinya *user*, maka interaktivitas akan bertambah.

Untuk mengenali wajah, dibutuhkannya model. Model dilatih dengan menggunakan *dataset*. Semakin besar *dataset* yang digunakan, maka tingkat akurasi dari model akan cenderung bertambah. Tetapi, dalam melatih *dataset* dibutuhkan waktu yang relatif lama sehingga sebagai alternatif dapat digunakan *pre-trained* model, yaitu model yang sudah dilatih sehingga siap untuk dipakai. Salah satu *pre-trained* model yang dapat digunakan adalah *pre-trained* model yang telah dilatih dengan *dataset* VGGFace2 yang disediakan oleh *library* FaceNet. Karena CNN hanya membutuhkan wajah dari gambar yang diambil sebagai

pelatihan, dibutuhkannya sebuah metode untuk mengambil gambar wajah saja dari gambar yang telah diambil. Salah satu algoritma yang dapat dipakai adalah MTCNN. Seiring dengan bertambahnya *user*, maka waktu pelatihan akan meningkat. Oleh karena itu, dibutuhkannya sebuah algoritma untuk membatasi dan menggantikan user yang sudah lama tidak menggunakan Jacob Salah satu algoritma yang dapat dipakai adalah Least Recently Used untuk menggantikan *user* yang tidak memakai Jacob dengan waktu yang paling lama.

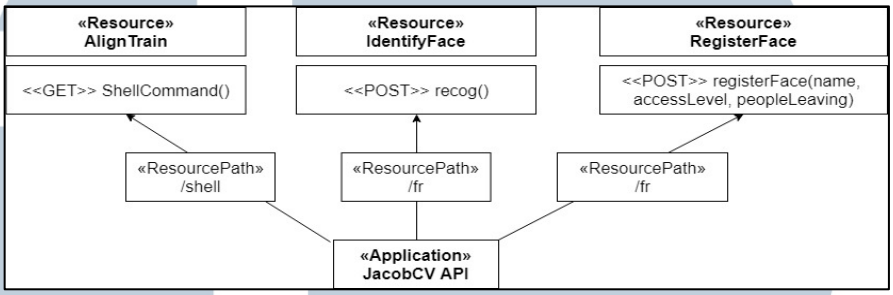
Ketiga, integrasi dengan *chatbot* Jacob. Vision dirancang sebagai pengembangan untuk *chatbot* Jacob. Mempertimbangkan bahwa *chatbot* Jacob dibangun menggunakan teknologi *web*, maka pembangunan Vision adalah dalam bentuk *web service* dengan arsitektur *Client Server* menggunakan FaceNet sebagai model utama dalam mengenali wajah. Model ini akan berjalan di server dengan bantuan *Application Programming Interface* (API). API pada Vision dibuat berbasis *web* menggunakan *framework* Flask 1.0.2 dengan bahasa pemrograman Python dan penyimpanan data pada *file system* di server. Selain dalam berkomunikasi, diperlukannya tampilan antarmuka yang mendukung sehingga keberadaan Vision dapat dengan mudah dilihat oleh pengguna tetapi tidak menutupi bagian dari tampilan antarmuka Jacob.

3.3. Perancangan Vision

Dari hasil analisis yang telah dilakukan, maka dirancanglah Vision dengan fungsionalitas yang diuraikan dengan diagram-diagram, yaitu *web service* model, diagram alur kerja, dan *Unified Modeling Language* (UML). Selain fungsionalitas, rancangan tampilan antarmuka integrasi akan dilakukan.

3.3.1. Web Service Model Vision

Secara umum, Vision dibuat sebagai *web service* yang akan berjalan ketika menerima *request* dari *client* untuk memproses data yang masuk. *Web service* yang akan dibangun dijabarkan dengan gambar di bawah ini.



Gambar 3.1 *Web service model* dari Vision

Web service yang dibangun terbagi menjadi 3 *resource*, yaitu *IdentifyFace* yang memiliki fungsi untuk melakukan identifikasi wajah, *RegisterFace* yang memiliki fungsi untuk mendaftarkan wajah baru, dan *AlignTrain* berfungsi untuk melakukan pemerataan dan pemotongan gambar serta melalukan pelatihan *classifier*. Setiap *resource path* dapat diakses dengan protokol HTTP dengan metode tertentu yang diizinkan. Dikarenakan terdapat 2 *resource* yang menggunakan *resource path* yang sama, maka apabila *request* diterima, penentu *resource* yang akan dijalankan adalah berdasarkan jumlah parameter data yang dikirimkan dari *client.* Bentuk data yang diterima maupun respon kembalian API adalah JSON.

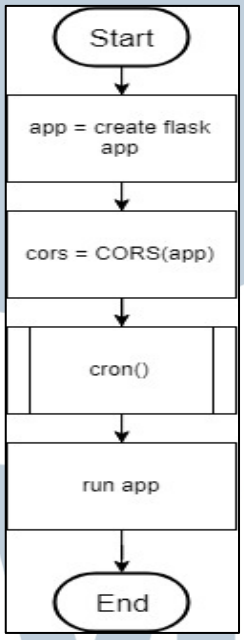
3.3.2. Alur Kerja Modul Vision

Alur kerja Vision berupa *back-end* yang diimplementasikan pada *web service* (API). *Web service* yang diimplementasikan berupa registrasi pengguna baru (admin dan *user*) dan pengenalan wajah. Fungsi pendukung yang dibangun untuk

menunjang fungsionalitas Vision seperti fungsi pemerataan dan pemotongan gambar serta pelatihan *classifier*. Selain itu, terdapat juga fungsi yang dibuat secara *multi-threading* untuk melakukan pengecekan data yang mungkin tidak sempurna pada saat pendaftaran dan akan dihapus.

A. Back-end

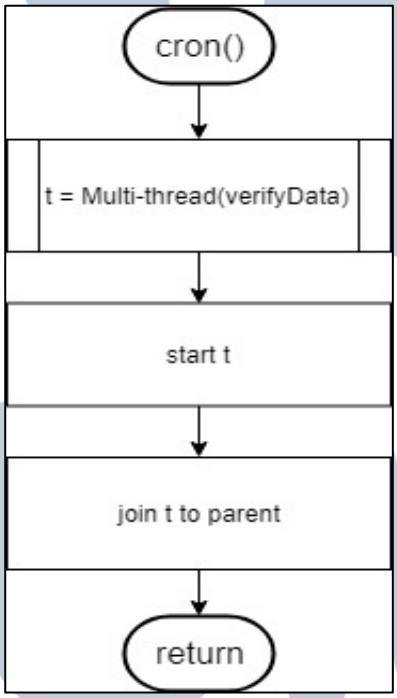
Web service ini rancang menggunakan framework Flask 1.0.2 dengan basis bahasa pemrograman Python. *Web service* menggambarkan alur kerja Vision secara kesluruhan dan dapat dilihat pada Gambar 3.2.



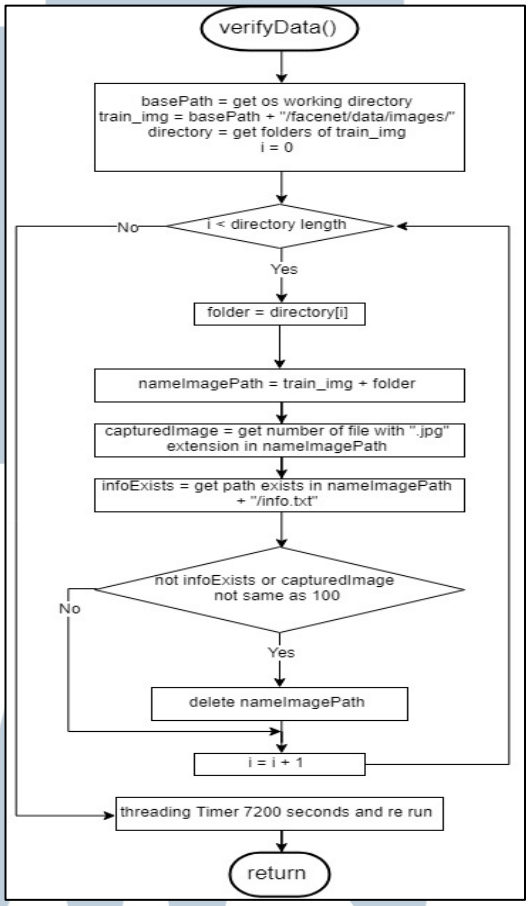
Gambar 3.2 Alur kerja *web service*

Sebelum *web service* dijalankan, terlebih dahulu dilakukan *multi-threading* dengan fungsi *cron* sebagai penunjang fungsionalitas Vision. Tujuan dari dilakukannya *multi-threading* pada Vision adalah agar terciptanya sebuah *thread* baru yang melakukan pekerjaan yang berulang tetap tidak melakukan pemblokiran terhadap pekerjaan lain yang akan dilakukan sehingga dapat melakukan secara bersamaan.

Proses cron kemudian menjalankan fungsi verifyData untuk melakukan pengecekan data pengguna di penyimpanan Vision dengan mengandalkan *Timer* dari *threading* sehingga akan menjalankan kembali fungsi yang sama setiap 7200 detik atau 2 jam. Apabila data yang tersimpan tidak memenuhi kriteria sebagai pengguna yang terdaftar, maka data tersebut akan dihapus. Proses cron dijabarkan pada Gambar 3.3 dan verifyData pada Gambar 3.4.



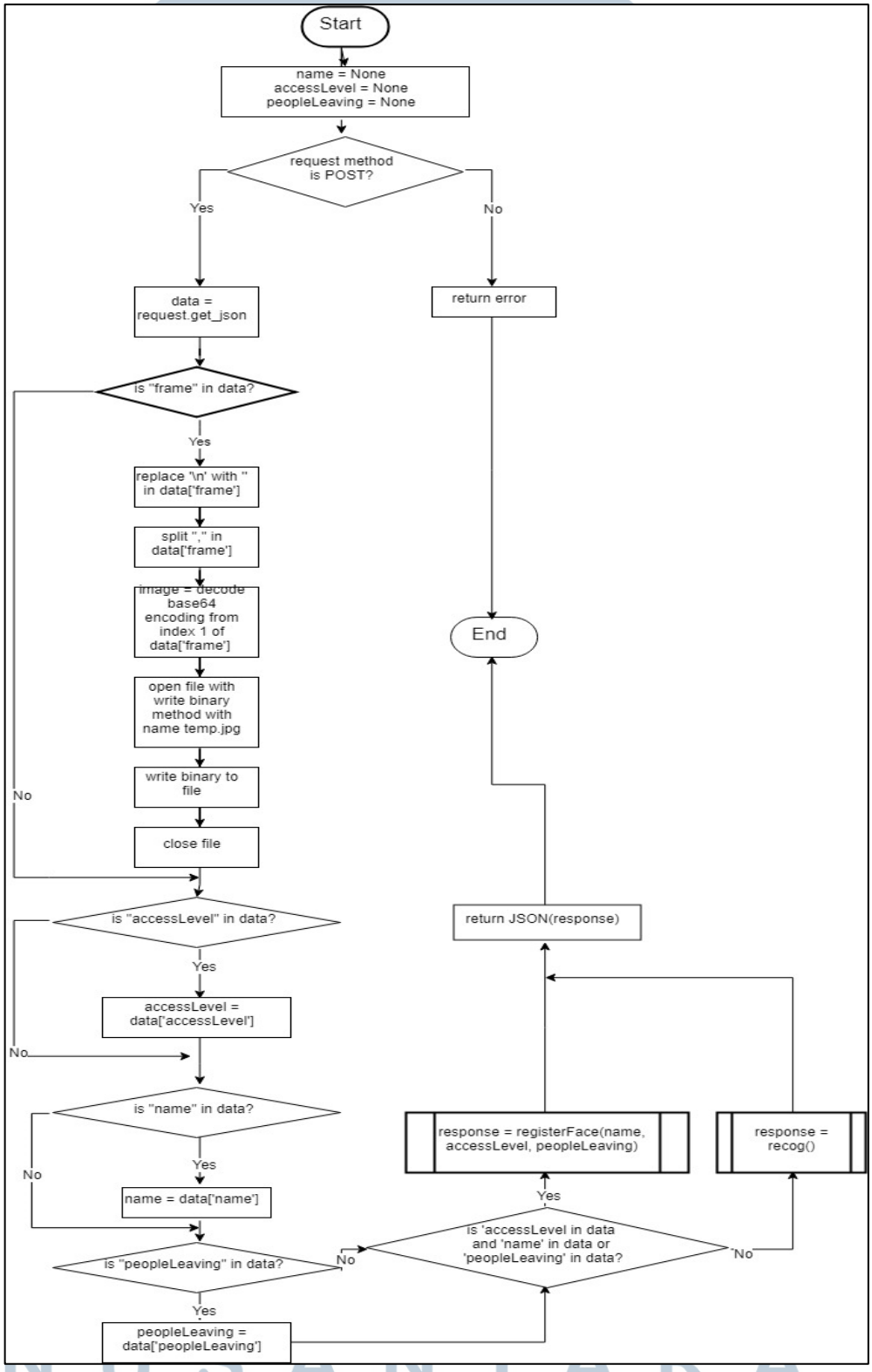
Gambar 3.3 Alur kerja proses cron



Gambar 3.4 Alur kerja verifyData

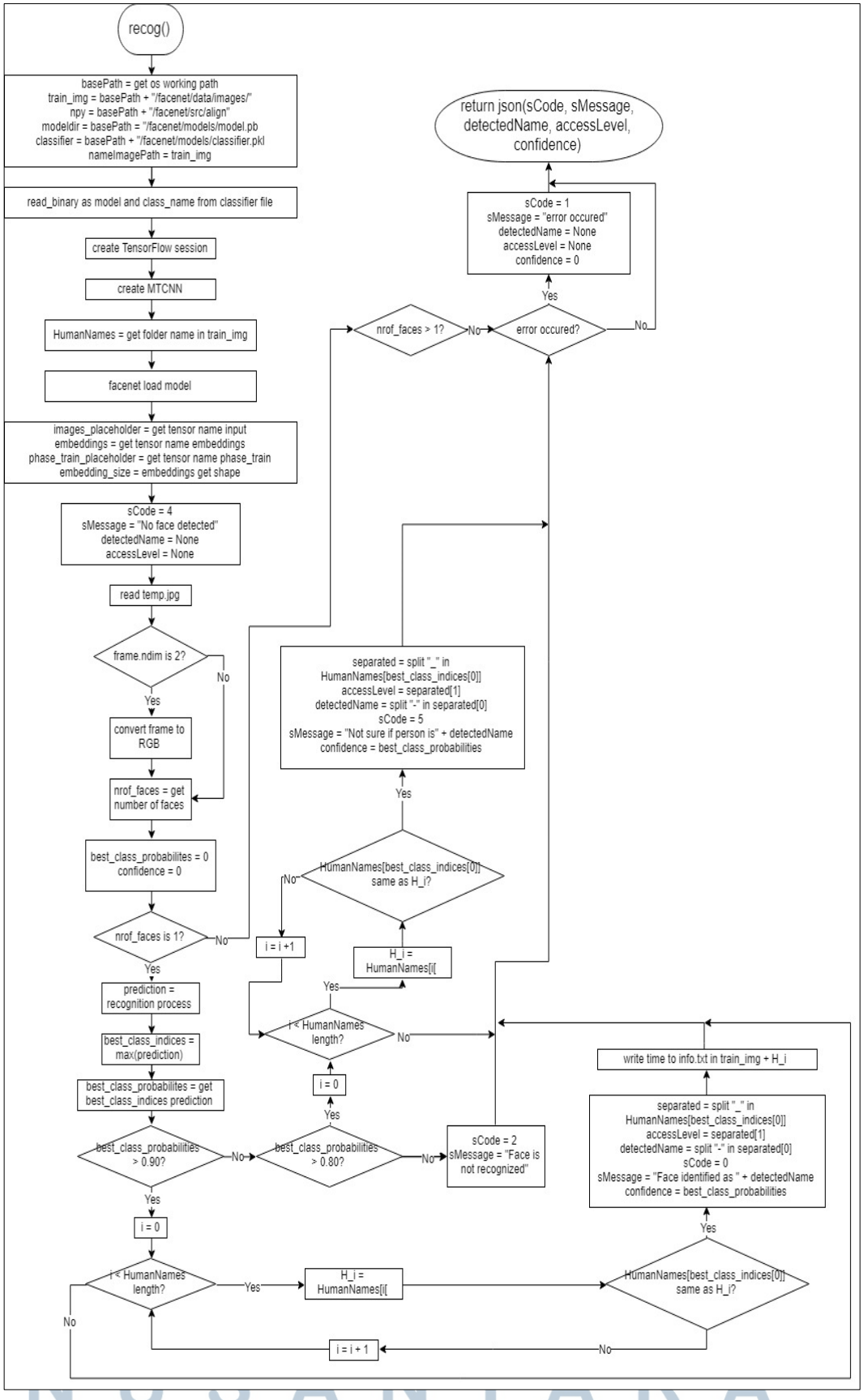
Setelah *web service* berjalan, *request* yang diterima oleh Vision dalam melakukan identifikasi wajah dan pendaftaran wajah adalah melalui API. Pada Gambar 3.5 dijabarkan alur kerja untuk *resource* IdentifyFace dan RegisterFace

pada API sesuai dengan *web service model* yang dapat diakses melalui HTTP protokol /fr.

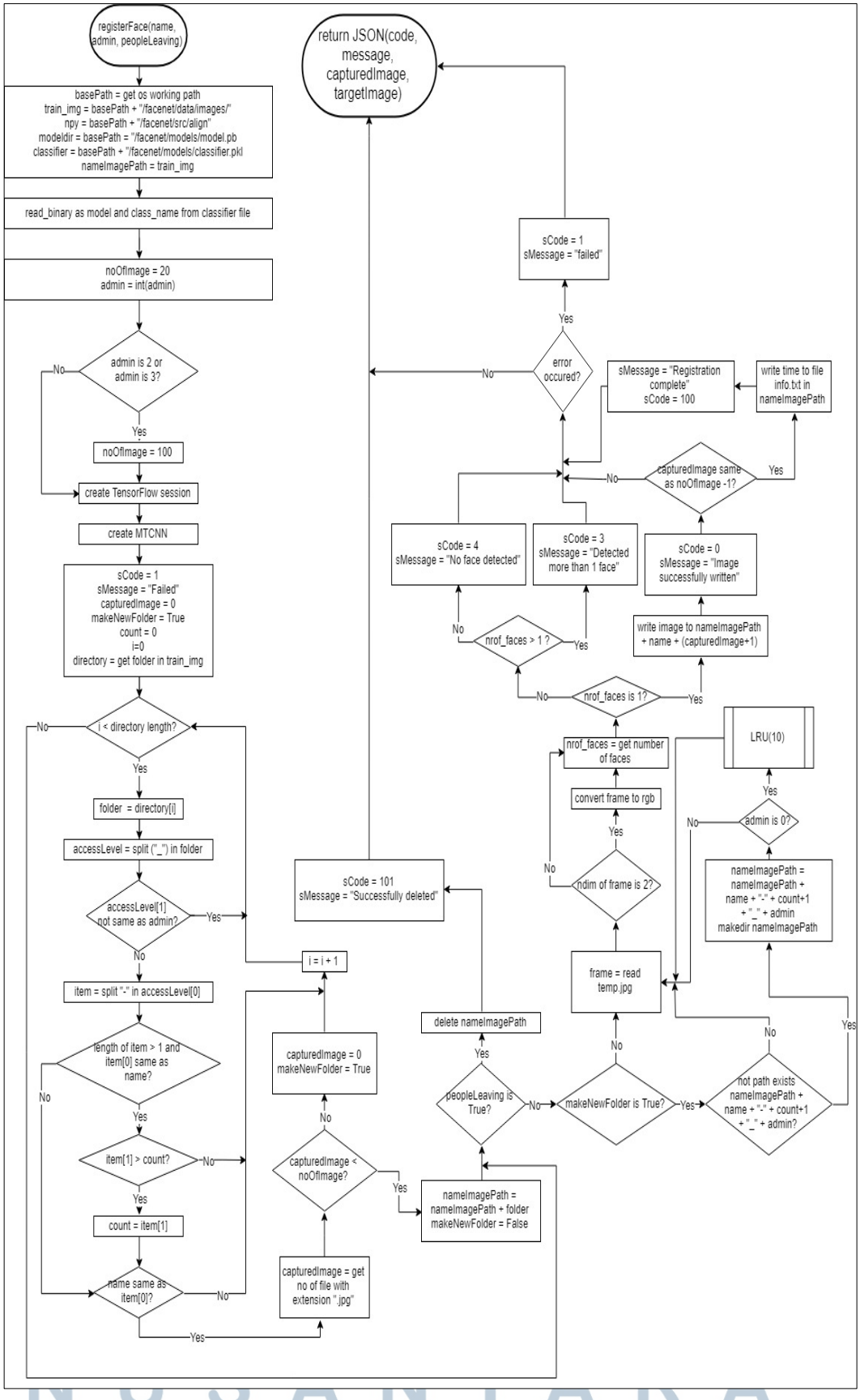


Gambar 3.5 Alur kerja API untuk *resource* IdentifyFace dan RegisterFace

Apabila *request* yang dikirimkan oleh *client* dikategorikan sebagai pengenalan wajah, maka alur kerja akan dialihkan kepada proses pengenalan wajah yaitu fungsi recog yang dapat dilihat pada Gambar 3.6. Vision dirancang dengan memiliki 2 jenis *threshold*, yaitu pada nilai 0.9 dan 0.8. Jika diatas 0.9, maka Vision yakin dengan hasil identifikasinya, tetapi apabila berada diatas nilai 0.8 dan dibawah 0.9, maka Vision tidak begitu yakin dengan hasil identifikasinya. Hal ini dapat dimanfaatkan pada saat integrasi agar dapat memberikan langkah penanggulangan yang tepat. Apabila *request* yang dikirimkan oleh *client* dikategorikan sebagai pendaftaran wajah, maka alur kerja akan dialihkan kepada proses pendaftaran wajah yang dapat dilihat pada Gambar 3.7.

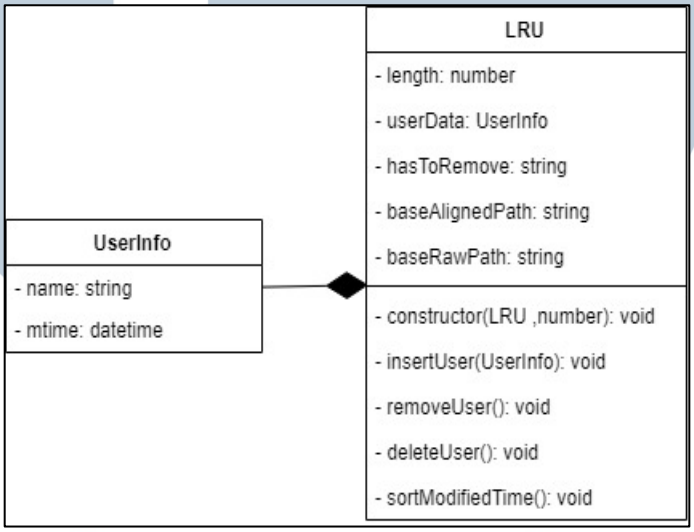


Gambar 3.6 Alur kerja recog



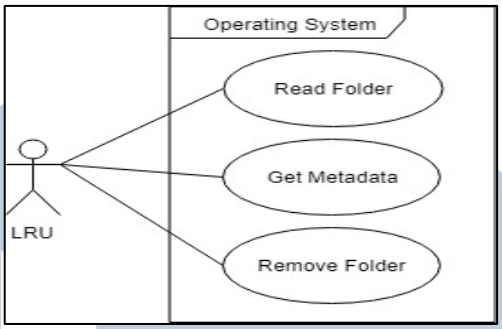
Gambar 3.7 Alur kerja registerFace

Didalam alur kerja pendaftaran, terdapat algoritma *Least Recently Used* (LRU) yang bertujuan untuk menghapus *user* yang sudah melebihi batas yang telah ditetapkan sesuai dengan rancangan yaitu 10 orang. *User* yang akan dihapus adalah *user* yang memenuhi kriteria, yaitu berdasarkan waktu aktif yang paling lama. Pada Gambar 3.8 sampai dengan Gambar 3.11 dijabarkan UML LRU yang diimplementasikan secara *Object Oriented Programming* (OOP), yaitu *class* diagram, *activity* diagram, *use case* diagram, dan *sequence* diagram.



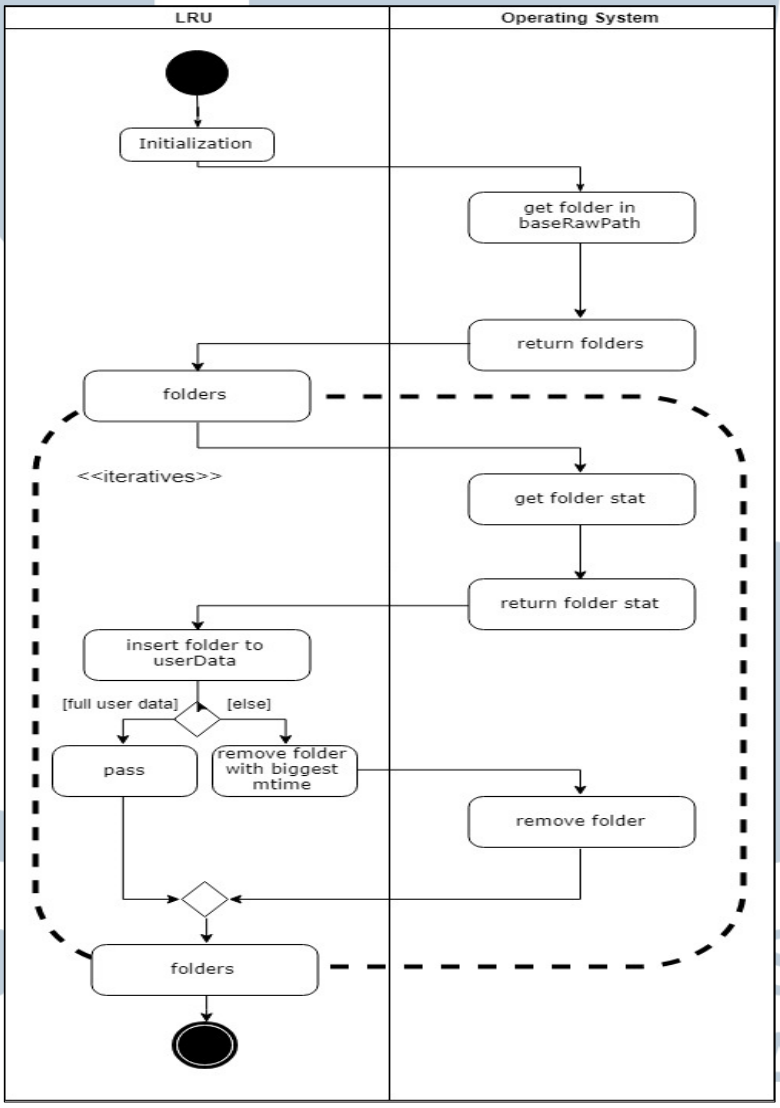
Gambar 3.8 *Class* diagram LRU

Class diagram pada Gambar 3.8 menunjukkan hubungan antar kelas yang terdapat pada algoritma LRU yang akan diimplementasikan. Class LRU terdiri dari 5 atribut dan 5 metode yang bersifat *private*. Atribut LRU dengan nama userData berisikan objek *class* UserInfo yang digunakan sebagai wadah untuk menampung informasi data yang akan dimasukkan kedalam *class* LRU untuk diolah. *Use case* pada Gambar 3.9 menunjukkan class *LRU* yang berinteraksi dengan sistem operasi. LRU dapat melakukan pembacaan *folder*, mengambil metadata *folder*, dan menghapus *folder* melalui sistem operasi.



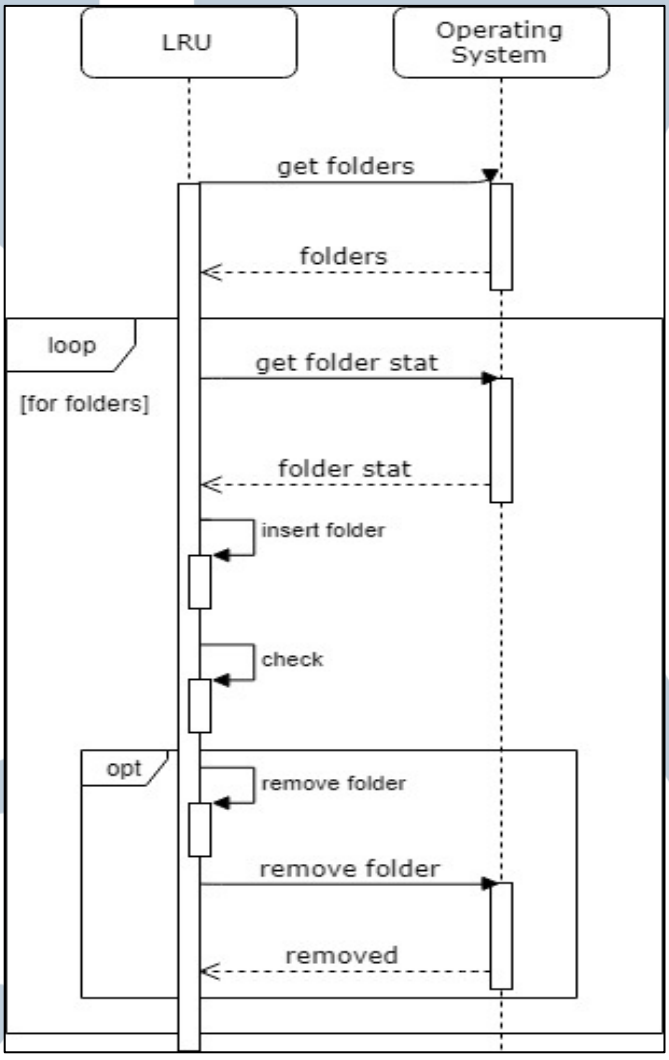
Gambar 3.9 *Use case* diagram LRU

Proses pengolahan data dalam *class* LRU diuraikan dalam Gambar 3.10.



Gambar 3.10 *Activity* diagram LRU

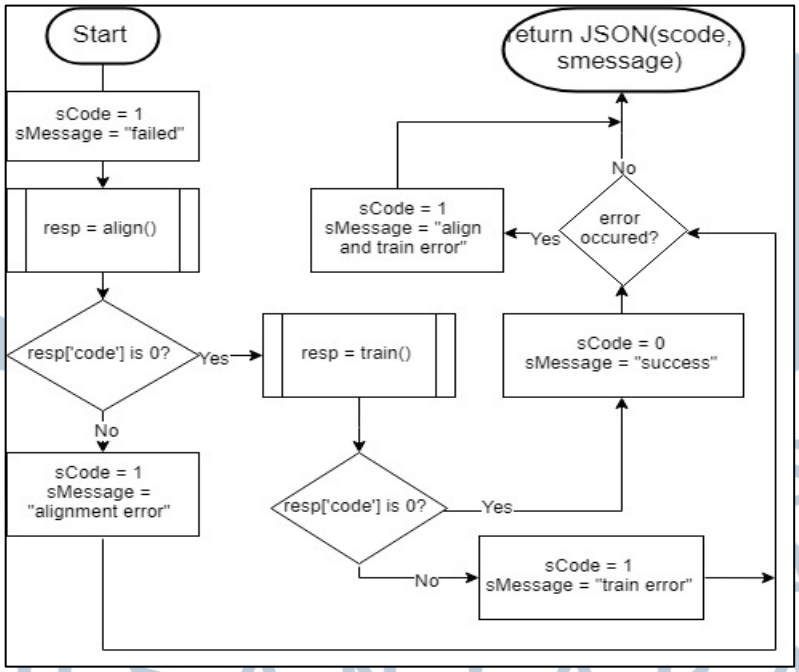
Metode *constructor* akan melakukan inisiasi untuk mengambil seluruh *folder* yang ada pada *path* yang tersedia pada atribut. Untuk setiap *folder* yang ada, *metadata* akan diambil dan *metadata* yang dipakai adalah *modified time*. Kemudian, *folder* tersebut akan dimasukkan kedalam *class* LRU berupa nama *folder* dan *modified time*. Apabila isi dari atribut userData telah melebihi atribut length dalam *class* LRU, maka akan dilakukan pengecekan terhadap isi dari userData. *Folder* yang akan dihapus dari userData maupun dari *file system* adalah folder dengan *modified time* paling lama.



Gambar 3.11 *Sequence* diagram *class* LRU

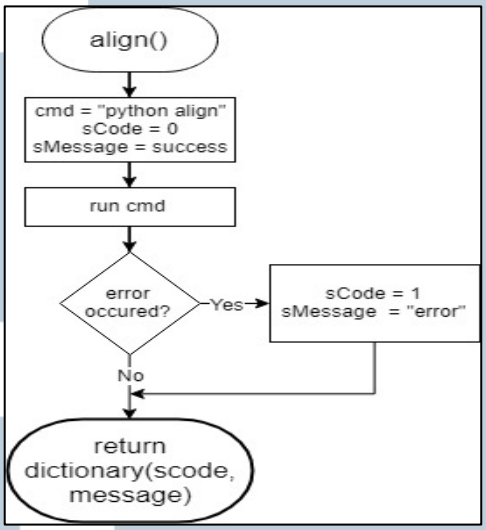
Gambar 3.11 menunjukkan *sequence* diagram, yaitu interaksi dari *class* LRU dengan sistem operasi dan *file system* dalam melakukan pengambilan *folder*, pengambilan *metadata folder*, dan juga penghapusan *folder*. *Class* LRU akan meminta sistem operasi untuk mengambil folder pada file system. Setelah mendapatkan seluruh *folder*, maka untuk setiap *folder* akan diambil *metadata*-nya dan kemudian dimasukkan kedalam userData dalam *class* LRU. Class LRU akan melakukan cek untuk setiap *folder* yang dimasukkan. Apabila userData pada *class* LRU telah penuh, maka *folder* yang akan dihapus dari userData adalah folder dengan *modified time* paling lama. Setelah menghapus dari userData, class LRU akan meminta sistem operasi untuk menghapus *folder* tersebut dari *file system*.

Setelah proses pendaftaran selesai, API yang disiapkan untuk melakukan pemerataan dan pemotongan gambar serta pelatihan *classifier* dengan *resource* bernama AlignTrain yang dapat diakses melalui protokol HTTP /shell dengan metode GET. Alur kerja *resource* AlignTrain dapat dilihat pada Gambar 3.12.



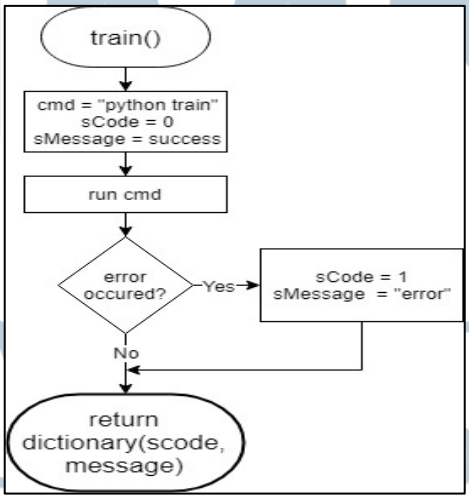
Gambar 3.12 Alur kerja *resource* AlignTrain

Dalam *resource* AlignTrain, terdapat 2 subproses yang digunakan sebagai pemerataan dan pemotongan gambar, yaitu *align* dan pelatihan classifier, yaitu *train*. Subproses *align* memanfaatkan *command line interface* untuk mengeksekusi baris kode. Gambar 3.13 menjabarkan alur kerja dari subproses *align*.



Gambar 3.13 Alur kerja subproses *align*

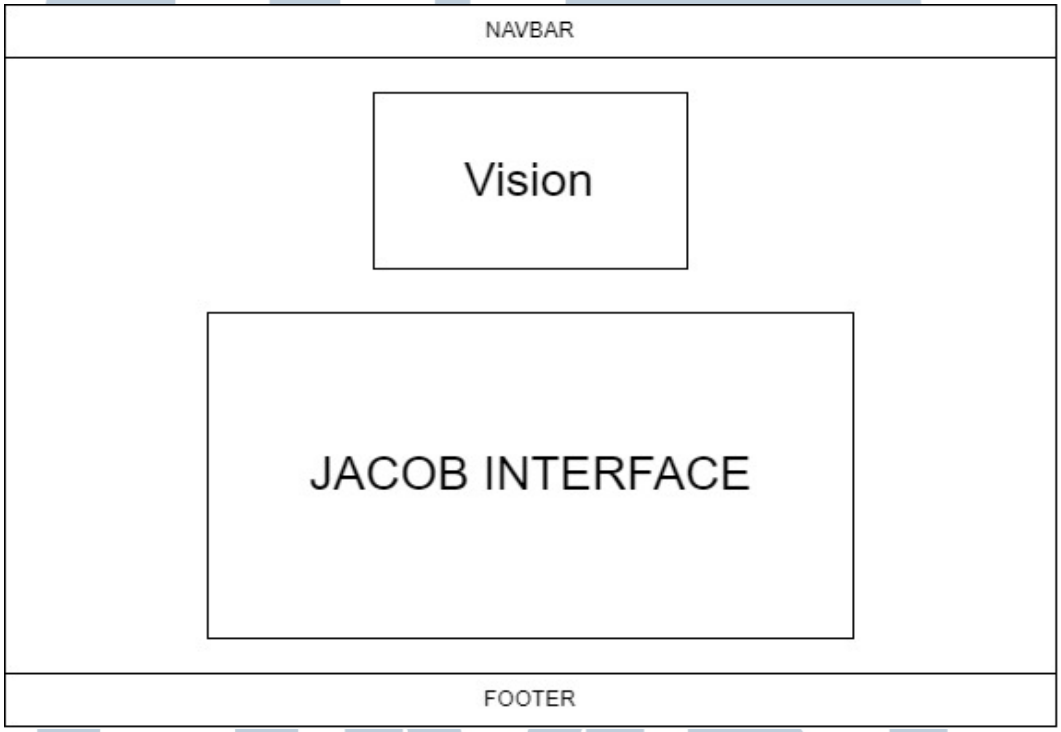
Sama seperti dengan subproses *align*, subproses *train* juga memanfaatkan *command line interface* untuk mengeksekusi baris kode. Alur kerja subproses *train* dapat dilihat pada Gambar 3.14.



Gambar 3.14 Alur kerja subproses *train*

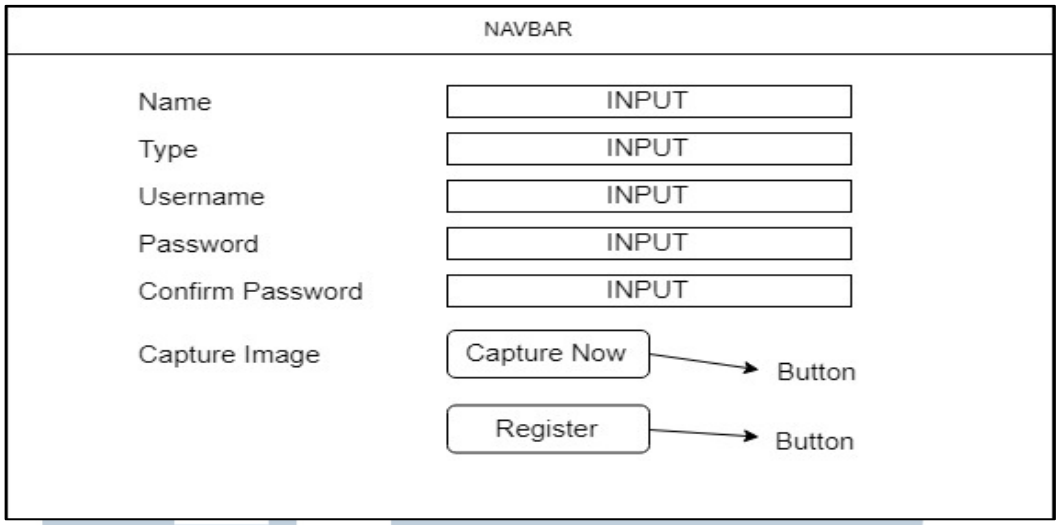
3.3.3. Rancangan Antarmuka Integrasi Vision

Setelah perancangan *back-end* Vision dilakukan, juga dibuat rancangan antarmuka Vision yang diintegrasikan dengan Jacob. Rancangan antarmuka dibentuk sedemikian rupa untuk mempermudah pengguna dalam menggunakan Jacob dengan tampilan yang sederhana. Rancangan antarmuka terbagi menjadi dua, yaitu untuk *user* dan admin.



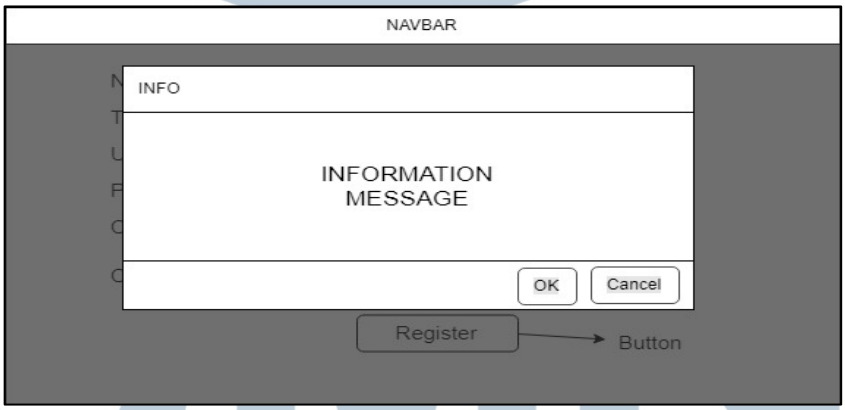
Gambar 3.15 Halaman awal pada chatbot Jacob

Gambar 3.15 menunjukkan rancangan antarmuka Vision untuk *user*. Vision akan menempati bagian atas dari halaman. Vision yang akan ditampilkan berupa gambar yang diambil secara langsung melalui kamera yang diakses melalui *browser*. Vision dirancang untuk tidak mengambil tempat yang terlalu besar karena Vision yang ditampilkan pada halaman awal hanya untuk memberikan tanda kepada pengguna. Selain pada halaman awal yang digunakan untuk *user*, antarmuka Vision juga dirancang pada halaman registrasi admin.



Gambar 3.16 Halaman registrasi admin

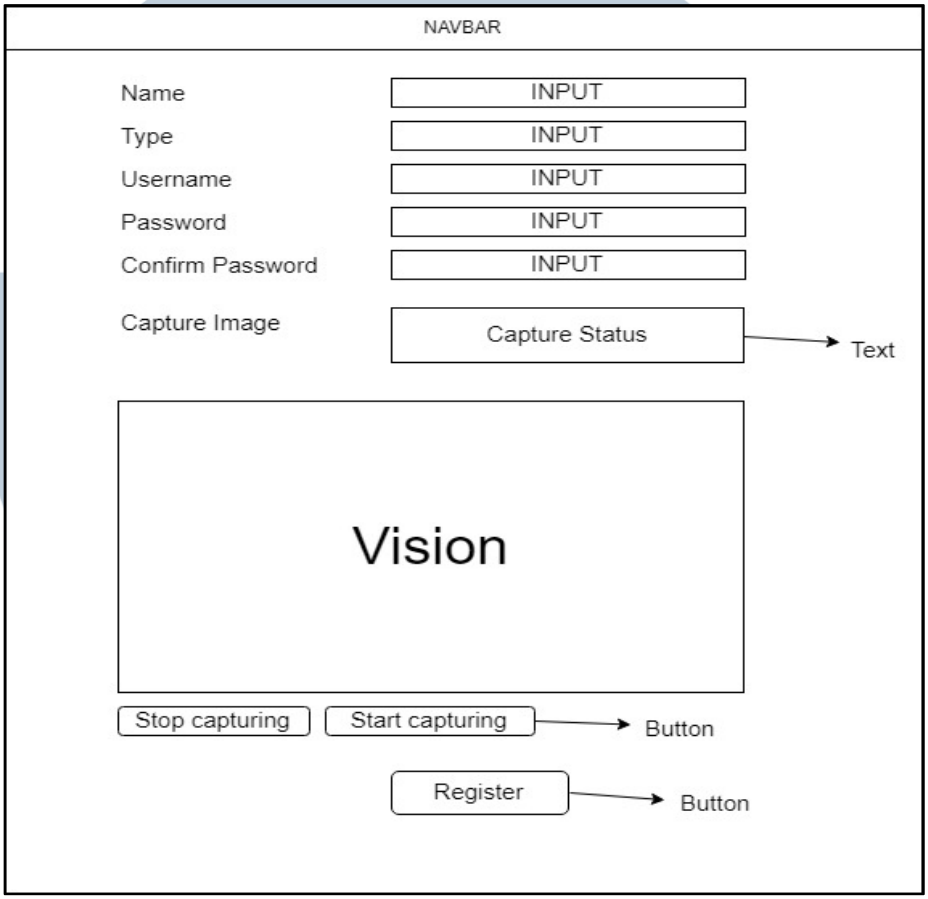
Gambar 3.16 menunjukkan rancangan antarmuka pada halaman registrasi admin. Apabila tombol *capture now* ditekan, maka akan memunculkan sebuah *modal* yang berisikan panduan dan informasi dalam sebelum proses pengambilan gambar dimulai.



Gambar 3.17 Tampilan *modal* panduan dan informasi

Gambar 3.17 menunjukkan rancangan tampilan untuk modal panduan dan informasi. Apabila tombol OK ditekan, maka modal akan ditutup dan tampilan akan berubah seperti pada Gambar 3.18. Vision akan muncul pada bagian bawah, tombol *capture now* akan berubah menjadi teks yang berisikan status pendaftaran. Terdapat

pula dua tombol yang berfungsi untuk memberhentikan sementara dan melanjutkan proses pengambilan gambar.



Gambar 3.18 Tampilan setelah tombol OK ditekan