PERANCANGAN SISTEM APLIKASI KLIK CHAT BISNIS PADA PT. SURYA LIFETIME INTERNATIONAL

KERJA PRAKTIK

Program Studi

S1 Sistem Informasi

Oleh :

AMRIZAL RIZKY FAJAR

17.41010.0110

FAKULTAS TEKNOLOGI DAN INFORMATIKA

UNIVERSITAS DINAMIKA

2020

PERANCANGAN SISTEM APLIKASI KLIK CHAT BISNIS PADA PT. SURYA LIFETIME INTERNATIONAL

Diajukan sebagian salah satu syarat untuk menyelesaikan

Program Sarjana Komputer

Disusun Oleh:

Nama **: AMRIZAL RIZKY FAJAR**

NIM **: 17410100110**

Program **: S1 (Strata Satu)**

Jurusan **: Sistem Informasi**

FAKULTAS TEKNOLOGI DAN INFORMATIKA

UNIVERSITAS DINAMIKA

2020

LEMBAR PENGESAHAN

PERANCANGAN SISTEM APLIKASI KLIK CHAT BISNIS PADA PT. SURYA LIFETIME INTERNATIONAL

Laporan Kerja Praktik oleh

Amrizal Rizky Fajar

Nim : 17410100110

Telah diperiksa, diuji dan disetujui

Surabaya, 20 Juli 2020

Disetujui :

Pembimbing	Penyelia
Digitally signed by Endra Rahmawati DN: cn=Endra Rahmawati, o, ou, email=rahmawati@dinamika.ac.id, c=ID Date: 2020.07.23 15:16:46 +07'00'	
Endra Rahmawati, M.Kom.	Firman Zain Syaifuddin
NIDN. 0712108701	

Mengetahui,

Ketua Program Studi S1 Sistem Informasi

Anjik Sukmaaji

Digitally signed by Anjik Sukmaaji
Date: 2020.07.24 08:02:22 +07'00'

Dr. Anjik Sukmaaji, S.Kom., M.Eng.

NIDN. 0731057301

SURAT PERNYATAAN

PERSETUJUAN PUBLIKASI DAN KEASLIAN KARYA ILMIAH

Sebagai mahasiswa Universitas Dinamika, saya :

Nama	:	Amrizal Rizky Fajar
Nim	:	17410100110
Program Studi	:	S1 Sistem Informasi
Fakultas	:	Fakultas Teknologi dan Informatika
Jenis Karya	:	Laporan Kerja Praktek
Judul Karya	:	**PERANCANGAN SISTEM APLIKASI KLIK CHAT BISNIS PADA PT. SURYA *LIFETIME INTERNATIONAL***

Menyatakan dengan sesungguhnya bahwa:

1. Demi pengembangan Ilmu Pengetahuan, Teknologi dan Seni, saya menyetujui memberikan kepada Universitas Dinamika Hak Bebas Royalti Non-Eksklusif *(Non-Exclusive Royalti Free Right)* atas seluruh isi/ sebagian karya ilmiah saya tersebut di atas untuk disimpan, dialihmediakan dan dikelola dalam bentuk pangkalan data *(database)* untuk selanjutnya didistribusikan atau dipublikasikan demi kepentingan akademis dengan tetap mencantumkan nama saya sebagai penulis atau pencipta dan sebagai pemilik Hak Cipta
2. Karya tersebut di atas adalah karya asli saya, bukan plagiat baik sebagian maupun keseluruhan, Kutipan karya atau pendapat orang lain yang ada dalam karya ilmiah ini adalah semata hanya rujukan yang dicantumkan dalam Daftar Pustaka saya
3. Apabila dikemudian hari ditemukan dan terbukti terdapat tindakan plagiat pada karya ilmiah ini, maka saya bersedia untuk menerima pencabut terhadap gelar kerjasama yang telah diberikan kepada saya.

Demikian surat pernyataan ini saya buat dengan sebenarnya.

Surabaya, 20 Juli 2020

Yang menyatakan

Amrizal Rizky Fajar.

NIM : 17410100110

ABSTRAK

PT. SURYA LIFETIME INTERNATIONAL sedang mengembangkan sebuah aplikasi bernama KLIK Chat Bisnis berbasis android. Masalah yang ditemui dalam pengembangan aplikasi KLIK Chat Bisnis ini sendiri ialah belum adanya sistem yang dapat mengintegrasikan *database* dengan perangkat android itu sendiri. Masalah ini dianggap cukup serius dikarenakan perangkat android yang sedang dikembangkan membutuhkan data yang ada pada server agar dapat berjalan dengan efektif.

Solusi dari masalah tersebut adalah perlunya pembuatan sebuah sistem yang dapat mengintegrasikan antara *database* dengan perangkat *mobile*. Tanpa adanya sistem ini perangkat *mobile* tidak memiliki sebuah sarana yang dapat memenuhi kebutuhan data dari perangkat *mobile* itu sendiri yang mengakibatkan tidak berjalannya fungsi yang terdapat pada aplikasi KLIK Chat Bisnis berbasis *mobile* tersebut.

Berdasarkan permasalahan yang ada maka penulis akan memberikan solusi yaitu berupa mengembangkan sebuah sistem yang dapat mengintegrasikan *database* yang ada pada server dengan perangkat android itu sendiri menggunakan *framework Laravel*. Karena *Laravel* dapat dikatakan sebagai *framework* yang sangat cocok dalam pengembangan sistem ini dan juga tingkat keamanan dalam membangan sistem menggunakan *framework* ini dapat dikatakan sangat baik.

Kata kunci: *web*, KLIK Chat, Laravel

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa atas segala limpahan rahmat yang diberikan sehingga penulis dapat menyelesaikan kerja praktik dan juga laporan dari kerja praktik. Laporan ini disusun berdasarkan kerja praktik dan hasil studi yang dilakukan selama hampir dua bulan di PT. SURYA LIFETIME INTERNATIONAL kota Surabaya.

Kerja Praktik ini membahas tentang pembuatan aplikasi KLIK Chat Bisnis pada PT SURYA LIFETIME INTERNATIONAL di kota Surabaya. Aplikasi KLIK Chat Bisnis sendiri merupakan kembangan dari aplikasi pertamanya dengan tujuan sebagai sarana bagi para pengusaha dalam memasarkan usahanya melalui ruang lingkup KLIK Chat.

Penyelesaian laporan kerja praktik ini tidak terlepas dari bantuan berbagai pihak yang telah memberikan banyak masukan, nasihat, saran, kritik dan dukungan moral maupun materil kepada penulis. Oleh karena itu penulis menyampaikan rasa terima kasih kepada:

1. Ayah dan ibuku tercinta serta keluarga besarku yang selalu mendoakan, mendukung, dan memberikan semangat di setiap langkah dan aktifitas penulis.
2. Bapak Prof. Dr. Budi Jatmiko, M.Pd. selaku rektor Institut Bisnis dan Informatika Stikom Surabaya yang telah mengesahkan dan memberikan kesempatan secara resmi dalam melakukan kerja praktik.
3. Bapak Dr. Anjik Sukmaaji, S.Kom., M.Eng selaku Kepala Program Studi Sistem Informasi Institut Bisnis dan Informatika Stikom Surabaya serta dosen

pembimbing dalam kegiatan kerja praktik yang telah memberikan izin kepada penulis untuk melakukan kerja praktik.

4. Bapak Fendy Mahatma selaku General Manager KLIK Chat (PT. SURYA LIFETIME INTERNATIONAL) yang telah memberikan dukungan serta kesempatan dalam melakukan kerja praktik kepada penulis.
5. Teman-teman tercinta yang memberikan bantuan dan dukungannya dalam penyusunan proposal ini.
6. Pihak-pihak lain yang tidak disebutkan satu-persatu yang telah memberikan bantuan dan dukungan kepada penulis.

Semoga Tuhan YME memberikan balasan yang setimpal kepada semua pihak yang telah memberikan bantuan, bimbingan, dan nasehat dalam proses kerja praktik ini.

Penulis menyadari bahwa kerja praktik ini yang dikerjakan masih banyak terdapat kekurangan, sehingga kritik yang bersifat membangun dan saran dari semua pihak sangatlah diharapkan agar aplikasi ini dapat diperbaiki menjadi lebih baik lagi dikemudian hari. Semoga laporan kerja praktik ini dapat diterima dan bermanfaat bagi penulis dan semua pihak.

Surabaya, 20 Juli 2020

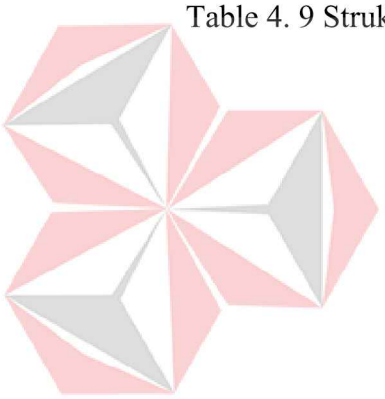
Penulis

DAFTAR ISI

Halaman

DAFTAR TABEL

Halaman



DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

BAB I

PENDAHULUAN

1.1 Latar Belakang

KLIK Chat merupakan anak perusahaan dari PT. SURYA LIFETIME INTERNATIONAL yang bergerak di bidang teknologi informasi dan komunikasi. Resmi berdiri pada tanggal 25 Maret 2011, KLIK Chat merupakan perusahaan penyedia layanan komunikasi berbagi pesan singkat dengan nama aplikasi yang sudah ada yaitu aplikasi KLIK Chat dan sedang dalam proses mengembangkan aplikasi KLIK Chat Bisnis.

Aplikasi KLIK Chat Bisnis merupakan aplikasi hasil pengembangan dari aplikasi KLIK Chat. Aplikasi KLIK Chat Bisnis merupakan aplikasi khusus diperuntukkan untuk para pengusaha. Aplikasi KLIK Chat Bisnis dibuat untuk membantu para pengusaha dalam memasarkan usahanya dalam ruang lingkup bisnis.

Saat ini aplikasi KLIK Chat Bisnis hanya berbasis *web* dan belum ada aplikasi berbasis *mobile*. Aplikasi berbasis *mobile* ini diperlukan karena dalam proses bisnis dari KLIK Chat itu sendiri lebih berfokus pada pemasaran aplikasi *mobile* nya.

Dengan kondisi yang terjadi saat ini, maka perlu adanya solusi untuk mengatasi masalah tersebut. Solusi dari permasalahan tersebut adalah perlu dibangun sistem aplikasi KLIK Chat Bisnis berbasis android dengan fitur-fitur yang diperlukan di android. Fitur – fitur tersebut antara lain pemantauan data perusahaan,

pemantauan kinerja karyawan pada perusahaan, interaksi dengan pelanggan melalui *chatting*, dan lain sebagainya.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang dijabarkan diatas, maka masalah pada kerja praktik ini adalah bagaimana merancang sistem aplikasi KLIK Chat Bisnis pada KLIK Chat Dapat dirumuskan sebagai berikut: Bagaimana merancang sistem aplikasi yang dapat mengintegrasikan antara *database* dengan aplikasi KLIK Chat Bisnis berbasis android

1.3 Batasan Masalah

Berdasarkan penjelasan pada latar belakang maka dibuatlah batasan masalah agar pembahasan masalah tidak melebar. Batasan masalah sebagai berikut:

a. Aplikasi berupa sistem aplikasi *backend*
b. Aplikasi dibangun menggunakan *framework* Laravel dan dengan *database MySQL*
c. Fitur dari aplikasi antara lain pemantauan data perusahaan, pemantauan kinerja karyawan, pembuatan sistem *ChatBot* oleh perusahaan, dan lain sebagainya.

1.4 Tujuan

Berdasarkan latar belakang dan rumusan masalah, maka tujuan dari kerja praktik ini adalah membangun sistem aplikasi KLIK Chat Bisnis pada perusahaan KLIK Chat.

1.5 Manfaat

Manfaat yang diharapkan dengan dibangunnya sistem aplikasi ini ialah sebagai berikut:

a. Membantu aplikasi KLIK Chat Bisnis berbasis *Website* agar dapat diakses dari android dan fitur-fitur tertentu dapat ditambahkan.

1.6 Sistematika Penulisan

Untuk memberikan gambaran menyeluruh terhadap masalah yang dibahas, maka sistematika penulisan dibagi ke dalam beberapa bab yaitu:

BAB I PENDAHULUAN

Pada bab ini menjelaskan tentang latar belakang dari hal-hal yang berhubungan dengan perusahaan, rumusan masalah, batasan masalah, tujuan yang ingin dicapai, manfaat yang diperoleh dengan adanya aplikasi yang telah dibuat, serta sistematika penulisan dari proposal.

BAB II GAMBARAN UMUM INSTANSI

Bab ini menjelaskan tentang KLIK Chat, mulai dari visi & misi perusahaan, dan stuktur organisasi.

BAB III LANDASAN TEORI

Pada bab ini membahas tentang teori-teori yang dianggap berhubungan dengan kerja praktik yang dilakukan, dimana teori-teori tersebut akan menjadi acuan untuk penyelesaian masalah.

BAB IV DESKRIPSI PEKERJAAN

Bab ini menguraikan tentang langkah-langkah dalam pengembangan sistem aplikasi yang akan digunakan untuk penyelesaian masalah yang membahas keseluruhan desain sistem pada PT. Surya Lifetime International

BAB V PENUTUP

Pada bab ini dibahas mengenai kesimpulan dan saran dari pembuatan sistem aplikasi KLIK Chat Bisnis untuk *platform mobile* pada PT. Surya Lifetime International terkait dengan tujuan dan permasalahan.

BAB II

GAMBARAN UMUM INSTANSI

2.1 Latar Belakang Perusahaan

Gambar 2. 1 Foto Perusahaan

KLIK Chat merupakan perusahaan yang bergerak pada bidang komunikasi pesan singkat yang terletak di kota Surabaya. Perusahaan ini merupakan salah satu anak perusahaan dari Lifetime (PT. Surya Lifetime International).

2.2 Identitas Instansi

Nama Istansi : KLIK Chat

Alamat : *Lifetime* Tower lt. 2 (Jl. Indragiri No. 36 Surabaya)

No. Telepon : (031) 99531010

Website : https://klikchat.com

Email : support@klikchat.com

2.3 Sejarah Perusahaan

KLIK Chat merupakan salah satu anak perusahaan dari PT. Surya Lifetime International yang bergerak di bidang komunikasi pesan singkat. Berdiri pada tanggal 29 Maret 2017, KLIK Chat dibentuk berdasar dari kebutuhan member Lifetime untuk saling berkomunikasi. Dalam mengembangkan fitur-fiturnya KLIK Chat menggunakan budaya Scrum, yaitu budaya yang digunakan oleh perusahaan perangkat lunak modern.

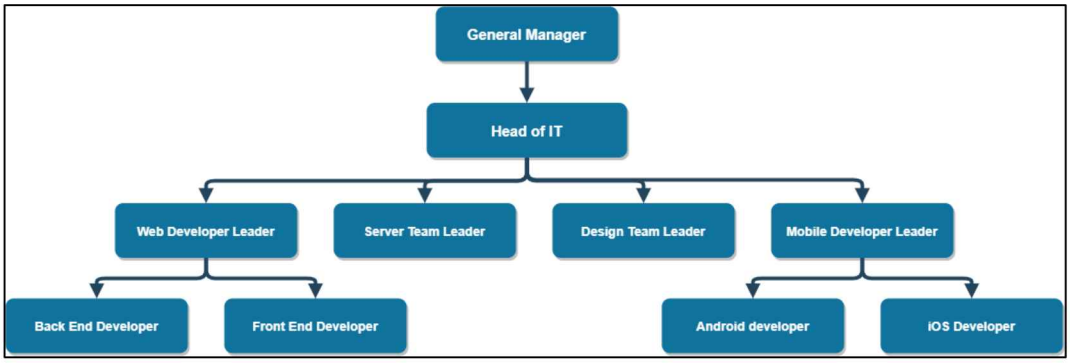
Budaya Scrum ini menjunjung tinggi *self management* serta transparansi planning maupun masalah dari seluruh pihak di perusahaan, sehingga produk dapat di-*deliver* lebih cepat ke pengguna. Produk pertamanya, yaitu KLIK Chat Android versi 1.0.0, di soft launching pada tanggal 25 Mei 2017, dua bulan setelah perusahaan ini dibentuk.

2.4 Jumlah Pengguna

Setelah *soft launching,* KLIK Chat memiliki jumlah pengguna *download* sebanyak 200 orang yang berlokasi di pulau jawa. Setelah satu tahun berdiri, KLIK Chat memiliki jumlah pengguna *download* sebanyak 500 orang. Hingga saat ini, KLIK Chat memiliki jumlah pengguna *download* sebanyak 700 orang dan akan terus bertambah.

2.5 Struktur Organisasi

Struktur Organisasi pada perusahaan KLIK Chat (PT. Surya Lifetime International) dapat dilihat pada Gambar 2.3 dibawah ini.



Gambar 2. 2 Stuktur Perusahaan

General Manager : Berkomunikasi dengan pemilik perusahaan mengenai renaca bisnis kedepan, berdiskusi dengan *Head of IT* mengenai rencana bisnis yang memungkinkan untuk segera diimplementasi, serta memastikan bisnis berjalan sesuai dengan rencana.

Head of IT : Riset mengenai teknologi terbaru, berdiskusi dengan *leader-leader* mengenai implementasi dari teknologi baru serta memastikan aplikasi KLIK Chat seluruhnya berjalan dengan baik.

Web Developer Leader : Memastikan aplikasi KLIK Chat berbasis *Website* berjalan dengan baik serta memastikan fungsi-fungsi yang dibutuhkan aplikasi KLIK

		Chat berbasis Android dan IOS pada sisi server berjalan dengan baik.
Back End Developer	:	Membangun dan mengembangkan fungsi-fungsi yang dibutuhkan oleh aplikasi KLIK Chat berbasis Android maupun IOS pada sisi server.
Front End Developer	:	Mengembangkan aplikasi KLIK Chat berbasis *Website*, serta seluruh yang berhubungan dengan *Website,* seperti *Website Company Profile.*
Mobile Developer Leader	:	Memastikan aplikasi KLIK Chat berbasis Android dan IOS berjalan dengan baik.
Android Developer	:	Mengembangkan aplikasi KLIK Chat berbasis Android.
IOS Developer	:	Mengembangkan aplikasi KLIK Chat berbasis IOS.

BAB III

LANDASAN TEORI

3.1 KLIK Chat

KLIK Chat adalah sebuah aplikasi yang berguna untuk memberikan pengguna alat untuk berkomunikasi dengan pengguna lain secara instan. Aplikasi tersebut mempunyai fungsi seperti *chating personal*, *chating group*, kirim gambar, *Voice Note*, kirim stiker, telfon antar pengguna, dan fungsi Video *Call* (KLIKChat, 2018).

3.2 *Website*

Website atau situs dapat diartikan sebagai kumpulan halaman-halaman yang digunakan untuk menampilkan informasi teks, gambar diam atau gerak, animasi, suara, dan atau gabungan dari semuanya, baik yang bersifat statis maupun dinamis yang membentuk satu rangkaian bangunan yang saling terkait, yang masing-masing dihubungkan dengan jaringan-jaringan halaman. Hubungan antara satu halaman *web* dengan halaman *web* yang lainnya disebut *Hyperlink*, sedangkan teks yang dijadikan media penghubung disebut *Hypertext*. Jenis-jenis *web* berdasarkan sifat atau *style*-nya (Hidayat, 2010)

Website Dinamis, merupakan sebuah *Website* yang menyediakan *content* atau isi yang selalu berubah-ubah setiap saat. Bahasa pemograman yang digunakan antara lain PHP, ASP, .NET dan memanfaatkan *database* MySQL atau MS SQL.

Misalnya *Website* www.artikel-it.com, www.detik.com, www.techno*mobile*.co.cc, www.polinpdg.ac.id dan lain-lain.

Website statis, merupakan *Website* yang content-nya sangat jarang diubah. Bahasa pemograman yang digunakan adalah HTML dan belum memanfaatkan *database*. Misalnya: *web* profile organisasi, dan lain-lain.

3.3 REST API

REST menurut Sinha et al dan Zhou et al merupakan gaya arsitektur dalam mendesain sebuah *web service* di mana desain REST memiliki *resource* yang dapat diakses melalui sebuah alamat HTTP URL yang *unique*. REST juga memungkinkan klien dapat melakukan *request* melalui *protokol* HTTP dengan mudah menggunakan URI seperti pada penelitian Kurniawan. Masing-masing alamat URL mengacu kepada kumpulan program yang akan dieksekusi dan akan mengembalikan pesan kepada pengirim perintah. (Rahman, 2013)

REST mengirimkan perintah yang akan dikerjakan oleh *server* menggunakan metode-metode HTTP *request method* yang disebut verb. Mengacu pada penelitian Lee dan Rahman terdapat delapan HTTP *request method*, yaitu *GET*, *POST*, *PUT*, *DELETE*, *OPTIONS*, *HEAD*,*TRACE*, dan *CONNECT*. Dalam penggunaan API REST hanya menggunakan empat dari metode-metode tersebut, yaitu: *GET*, *POST*, *PUT*, dan *DELETE*. Pesan yang diterima dari *server* berupa kode HTTP berhasil atau gagal di dalam header dan isi pesan hasil pengolahan program itu sendiri. Berikut adalah kode HTTP yang sering digunakan dalam penggunakan REST API

200 *OK*	:	Perintah yang dikirim ke *server* benar dan berhasil dijalankan.
400 *Bad Request*	:	Perintah yang dikirim ke *server* berisi isian yang salah.
401 *Unauthorized*	:	Pengirim perintah mengirimkan kode kunci yang salah.
403 *Forbidden*	:	Pengirim perintah tidak memiliki hak akses ke dalam *resource* yang dituju
404 *Not Found*	:	*Resource* yang dituju tidak ditemukan dalam *server*.
429 *Too Many Requests*	:	Pengirim perintah mengakses mencapai/melebihi dari *limit* yang telah ditentukan dari batas waktu tertentu.
500 *Internal Server Error*	:	*Server* atau potongan program dalam *resource* mengalami kesalahan

3.4 PHP

PHP atau kependekan dari *Hypertext Preprocessor* adalah salah satu bahasa pemrograman *open source* yang sangat cocok atau dikhususkan untuk pengembangan *web* dan dapat ditanamkan pada sebuah skripsi HTML. Bahasa PHP dapat dikatakan menggambarkan beberapa bahasa pemrograman seperti C, Java, dan Perl serta mudah untuk dipelajari. PHP merupakan bahasa *scripting server – side*, dimana pemrosesan datanya dilakukan pada sisi server. Sederhananya, server

lah yang akan menerjemahkan skrip program, baru kemudian hasilnya akan dikirim kepada *client* yang melakukan permintaan.

Adapun pengertian lain PHP adalah akronim dari *Hypertext Preprocessor*, yaitu suatu bahasa pemrograman berbasiskan kode – kode (*script*) yang digunakan untuk mengolah suatu data dan mengirimkannya kembali ke *web browser* menjadi kode HTML".

Pada prinsipnya server akan bekerja apabila ada permintaan dari client. Dalam hal ini *client* menggunakan kode-kode PHP untuk mengirimkan permintaan ke server. Sistem kerja dari PHP diawali dengan permintaan yang berasal dari halaman *Website* oleh *browser*. Berdasarkan URL atau alamat *Website* dalam jaringan internet, *browser* akan menemukan sebuah alamat dari *webserver*, mengidentifikasi halaman yang dikehendaki, dan menyampaikan segala informasi yang dibutuhkan oleh *webserver*.

Selanjutnya *webserver* akan mencarikan berkas yang diminta dan menampilkan isinya di *browser*. *Browser* yang mendapatkan isinya segera menerjemahkan kode HTML dan menampilkannya. Lalu bagaimana apabila yang dipanggil oleh *user* adalah halaman yang mengandung *script* PHP Pada prinsipnya sama dengan memanggil kode HTML, namun pada saat permintaan dikirim ke *web*-server, *web*-server akan memeriksa tipe file yang diminta *user*. Jika tipe file yang diminta adalah PHP, maka akan memeriksa isi *script* dari halaman PHP tersebut.

Apabila dalam file tersebut tidak mengandung *script* PHP, permintaan *user* akan langsung ditampilkan ke *browser*, namun jika dalam file tersebut mengandung *script* PHP, maka proses akan dilanjutkan ke modul PHP sebagai mesin yang

menerjemahkan *script*s PHP dan mengolah *script* tersebut, sehingga dapat dikonversikan ke kode-kode HTML lalu ditampilkan ke *browser user*. (Firman, Wowor, & Najoan, 2016)

3.5 Laravel

Menurut Otwell (2011), Laravel adalah kerangka kerja aplikasi *web* dengan *syntax* yang ekspresif dan elegan. Dengan laravel pengembangan menjadi pengalaman yang menyenangkan dan kreatif untuk benar-benar memuaskan. Laravel berusaha meringankan tugas-tugas umum yang digunakan di sebagian besar proyek *web*, seperti otentikasi, perutean, sesi, dan caching.

Laravel bertujuan untuk membuat proses pengembangan yang menyenangkan bagi pengembang tanpa mengorbankan fungsionalitas aplikasi. Untuk tujuan tersebut, Laravel telah berupaya menggabungkan yang terbaik dari apa yang telah ada dalam kerangka kerja *web* lain, termasuk kerangka kerja yang diterapkan dalam bahasa lain, seperti Ruby on Rails, ASP.NET MVC, dan Sinatra. (Otwell, 2020)

Laravel mudah untuk diakses, namun kuat, menyediakan alat-alat canggih yang dibutuhkan untuk aplikasi besar. Dengan inversi dari *control container* yang luar biasa, *migration system* yang ekspresif, dan dukungan pengujian unit terintegrasi yang ketat memberi pengembang alat yang dibutuhkan untuk membangun aplikasi apa pun yang akan digunakan untuk tugas.

Laravel adalah kerangka kerja pengembangan *web* dengan metode *Model-View-Controller* (MVC) yang ditulis dalam PHP. Laravel dirancang untuk

meningkatkan kualitas perangkat lunak dengan mengurangi biaya pengembangan awal dan biaya pemeliharaan berkelanjutan, dan untuk meningkatkan pengalaman bekerja dengan aplikasi, Laravel memberikan *syntax* yang ekspresif dan jelas dengan serangkaian fungsionalitas inti yang akan menghemat waktu implementasi. (McCool, 2012)

Laravel merupakan salah satu dari sedikit kerangka kerja PHP yang menawarkan modularitas kode sejati. Hal tersebut dicapai melalui kombinasi driver dan sistem *bundle*nya. Driver memungkinkan pengembang untuk dengan mudah mengubah dan memperluas fungsi caching, sesi, basis data, dan otentikasi. Dengan menggunakan *bundle*, pengembang dapat mengemas segala jenis kode untuk digunakan kembali sendiri atau untuk disediakan ke seluruh komunitas Laravel. Hal tersebut sangat berguna karena apa pun yang dapat ditulis dalam Laravel dapat dikemas sebagai *bundle*, dari perpustakaan sederhana ke seluruh aplikasi *web*. Situs *web bundle* Laravel memungkinkan pengembang untuk menelusuri *bundle* yang telah dibangun oleh komunitas serta untuk menampilkan *bundle* yang dikembangkan sendiri. Ini adalah sumber daya berharga dari perpustakaan dan subsistem pihak ketiga yang secara dramatis dapat memudahkan pengembangan aplikasi *web*.

3.6 MySQL

MySQL adalah sebuah "*SQL client*/ server *relational database management system*" yang berasal dari Scandinavia. Pada MySQL sudah termasuk SQL server,

program *client* untuk mengakses server, hal–hal yang berguna dalam hal administrasi, dan sebuah "*programming interface*" untuk menulis program sendiri.

MySQL bukan sebuah *project* yang *open source* karena dalam keadaan tertentu diperlukan "*license*". Tetapi kepopuleran dari MySQL terus berkembang dalam komunitas *open source* karena me-lisensikannya tidak terlalu sulit.

MySQL juga dapat berjalan pada personal komputer (banyak pengembangan dari MySQL terjadi pada *system* yang tidak mahal yaitu Linux *System*). Tetapi MySQL juga portable dan dapat berjalan pada sistem operasi yang komersial seperti misalnya Windows, Solaris, Irix. (Setiabudi, 2015)

MySQL menggunakan bahasa SQL. SQL (*Structured Query Langguage*) adalah bahasa standard yang digunakan untuk mengakses server *database*.

Beberapa keunggulan MySQL dibandingkan dengan *database* lain adalah:

Kecepatan

MySQL cepat. Para pengembang berpendapat bahwa MySQL adalah *database* yang tercepat yang didapat.

1. **Kemudahan**

   MySQL adalah simple *database system* dengan performa tinggi dan tidak kompleks untuk setup, dan administrator, dibanding dengan *system* yang lebih besar.

2. **Biaya**

   MySQL gratis untuk semua pengguna.

3. **Mendukung bahasa Query**

   MySQL memahami SQL, juga dapat mengakses MySQL menggunakan aplikasi yang mendukung ODBC

4. **Kemampuan**

   Banyak *client* dapat berhubungan dengan server pada saat yang bersamaan. Clients dapat menggunakan multiple *database* secara bersamaan.

3.8 *Blackbox Testing*

Menurut (Rangta, 2019), Pengujian *blackbox* didefinisikan sebagai teknik pengujian di mana fungsionalitas dari aplikasi yang diuji atau *the Application Under Test* (AUT), diuji tanpa melihat struktur kode internal, detail implementasi dan pengetahuan jalur internal perangkat lunak. Dalam Pengujian *Blackbox* kami hanya fokus pada *input* dan *output* dari sistem perangkat lunak tanpa melihat tentang pengetahuan internal dari program perangkat lunak.

Berikut adalah langkah-langkah umum yang diikuti untuk melakukan semua jenis Pengujian Kotak Hitam.

1) Awalnya, persyaratan dan spesifikasi sistem diperiksa.
2) *Tester* memilih *input* yang *valid* (skenario uji positif) untuk memeriksa apakah sistem yang akan diuji atau *system under test* (SUT) memprosesnya dengan benar. Juga, beberapa *input* yang tidak *valid* (skenario uji negatif) dipilih untuk memverifikasi bahwa SUT dapat mendeteksi mereka.
3) *Tester* menentukan *output* yang diharapkan untuk semua *input* tersebut.
4) Penguji perangkat lunak membuat kasus uji dengan *input* yang dipilih.

5) Kasus uji dieksekusi.
6) Penguji perangkat lunak membandingkan keluaran aktual dengan keluaran yang diharapkan.
7) Aplikasi belum dapat digunakan jika ada yang diperbaiki dan perlu diuji ulang.

Ada beberapa jenis *blackbox Testing*, berikut diantaranya yaitu:

1. Pengujian fungsional - Jenis *blackbox Testing* ini terkait dengan persyaratan fungsional suatu sistem; itu dilakukan oleh penguji perangkat lunak.
2. Pengujian non-fungsional - Jenis *blackbox Testing* ini tidak terkait dengan pengujian fungsionalitas tertentu, tetapi persyaratan non-fungsional seperti kinerja, skalabilitas, kegunaan.
3. Pengujian regresi - Pengujian Regresi dilakukan setelah perbaikan kode, peningkatan atau pemeliharaan sistem lainnya untuk memeriksa kode baru tidak memengaruhi kode yang ada.

Berikut ini adalah strategi uji yang banyak digunakan dalam *blackbox Testing:*

1. *Equivalence Class Testing*: Digunakan untuk meminimalkan jumlah kasus uji yang mungkin hingga tingkat optimal sambil mempertahankan cakupan uji yang wajar.
2. *Boundary Value Testing*: Pengujian nilai batas difokuskan pada nilai pada batas. Teknik ini menentukan apakah rentang nilai tertentu dapat diterima oleh sistem atau tidak. Ini sangat berguna dalam mengurangi jumlah kasus uji. Ini paling cocok untuk sistem di mana *input* berada dalam rentang tertentu.

3. *Decision Table Testing*: Tabel keputusan menempatkan penyebab dan efeknya dalam sebuah matriks. Ada kombinasi unik di setiap kolom.

BAB IV

DESKRIPSI PEKERJAAN

4.1 Analisis dan Desain Sistem

Berdasarkan hasil observasi dan wawancara di PT. Surya Lifetime International ditemukan permasalahan belum adanya sistem aplikasi yang dapat mengintegrasikan antara *database* dengan aplikasi KLIK Chat Bisnis berbasis android. Hal ini merupakan kebutuhan yang dianggap penting dikarenakan adanya sebuah kebutuhan data yang diperlukan agar aplikasi KLIK Chat Bisnis di *platform* android dapat berjalan susai dengan tujuan.

Sistem aplikasi untuk KLIK Chat Bisnis berbasis android ini menggunakan *framework* Laravel. *Framework* Laravel sendiri dianggap mudah dan digunakan dan juga sudah sangat sering digunakan di dunia kerja. Selain mudah digunakan, tujuan menggunakan *framework* ini yaitu agar dapat mudah dipelajari apabila ada perbuahan pada sistem di masa mendatang.

4.2 Identifikasi Pengguna

Berdasarkan hasil observasi dan wawancara pada PT. Surya Lifetime International maka dapat dilakukan identifikasi pengguna untuk sistem yang akan dibuat. Pengguna tersebut dapat diidentifikasikan, yaitu:

1. *Super Admin*
2. *Customer Service*
3. Karyawan

4.3 Identifikasi Data

Berdasarkan hasil observasi, wawancara dan juga identifikasi pengguna maka dapat dilakukan identifikasi data untuk sistem yang akan dibuat. Data tersebut dapat diidentifikasikan, yaitu:

1. Data *Master User*
2. Data *Master Company*
3. Data *Division*
4. Data *Chat Bot*
5. Data *Chat Default*
6. Data FAQ
7. Data Transaksi *Ticket Chat*

4.4 Identifikasi Kebutuhan Fungsional

Berdasarkan hasil observasi, wawancara, identifikasi pengguna dan juga identifikasi data maka dapat dilakukan identifikasi kebutuhan fungsional untuk sistem yang akan dibuat. Kebutuhan fungsional tersebut dapat diidentifikasikan, yaitu:

1. Fungsi *Master User*
2. Fungsi *Master Company*
3. Fungsi Pengelolaan *Division*
4. Fungsi Pengelolaan *Chat Bot*
5. Fungsi Pengelolaan *Chat Default*
6. Fungsi Pengelolaan FAQ

7. Fungsi Pengelolaan *Dashboard*
8. Fungsi Pengelolaan *Ticket Chat*

4.5 Analisis Kebutuhan Pengguna

1. Super Admin

Tugas dan tanggung jawab dari super admin sendiri yaitu melakukan pengelolaan terhadap setiap data baik *master* maupun transaksi.

Table 4. 1 Analisis Kebutuhan Pengguna Super Admin

Nama Pengguna	**Tugas dan Tanggung Jawab**	**Kebutuhan Data**	**Kebutuhan Informasi**	**Kebutuhan Dokumen / Output**
Super Admin	Melakukan pengelolaan terhadap data *master* dan juga data transaksi	1. Data *Master User* 2. Data *Master Company* 3. Data *Division* 4. Data *Chat Bot* 5. Data *Chat Default* 6. Data FAQ 7. Data *Ticket Chat*	1. Daftar *User* 2. Informasi *Company* 3. Daftar *Division* 4. Daftar *ChatBot* 5. Daftar *Chat Default* 6. Daftar FAQ 7. Daftar *Ticket Chat*	

2. *Customer Service*

Tugas dan tanggung jawab dari *Customer Service* yaitu melakukan pengelolaan terhadap fungsi pengelolaan *Ticket Chat*.

Table 4. 2 Analisis Kebutuhan Pengguna *Customer Service*

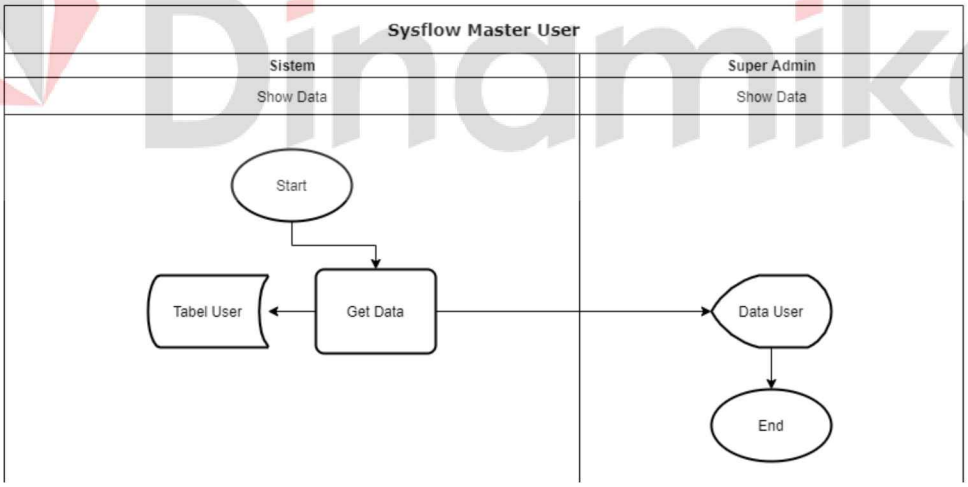
Nama Pengguna	**Tugas dan Tanggung Jawab**	**Kebutuhan Data**	**Kebutuhan Informasi**	**Kebutuhan Dokumen / Output**
Customer Service	Melakukan pengelolaan terhadap fungsi *Ticket Chat*	1. Data Transaksi *Ticket Chat*	1. Daftar *Ticket Chat*	

4.6 *Sysflow*

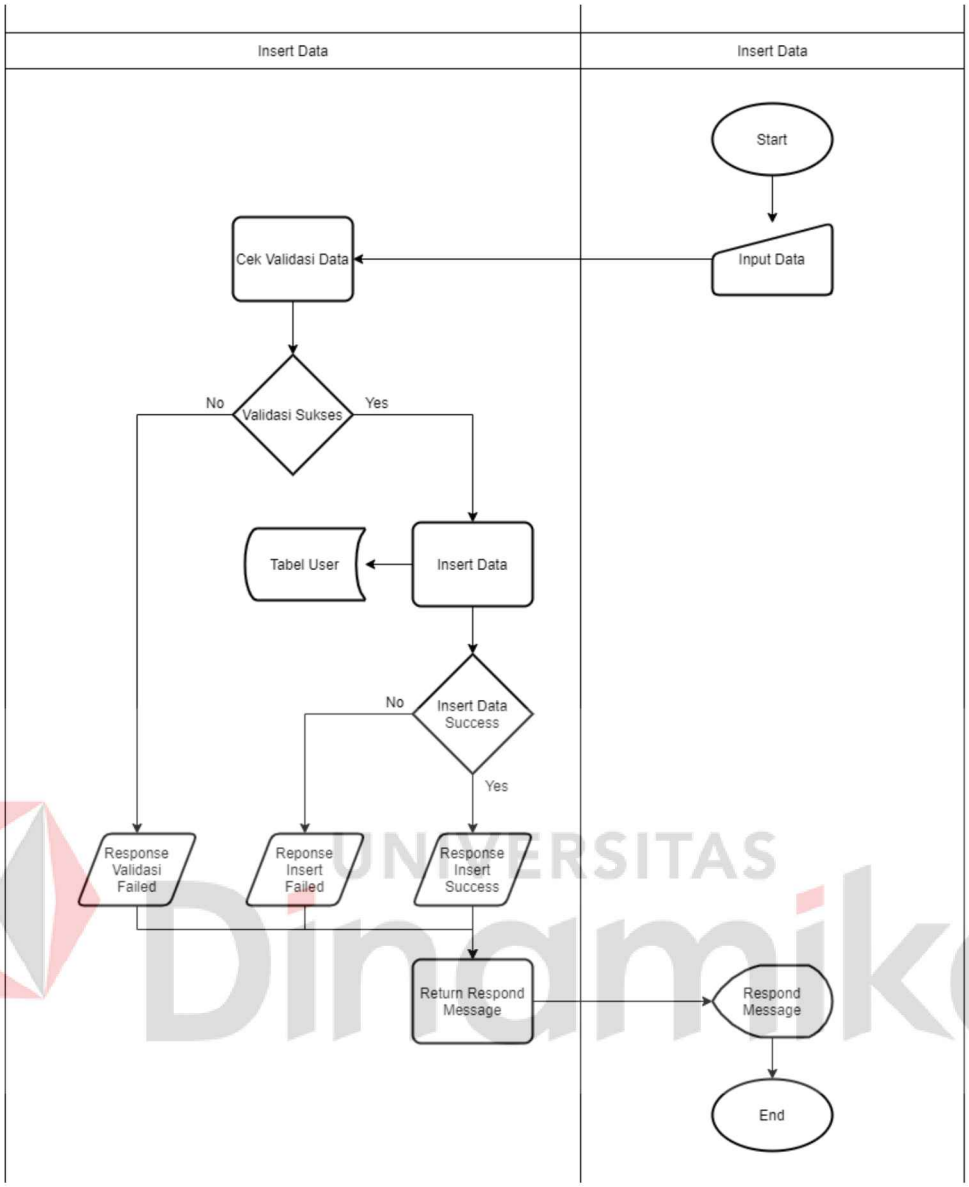
4.6.1 *Sysflow Master*

Sysflow master sendiri dibagi menjadi *Sysflow master user* dan juga *Sysflow master Company*. Berikut merupakan penjelasan dari *Sysflow master user* dan *Sysflow master Company*.

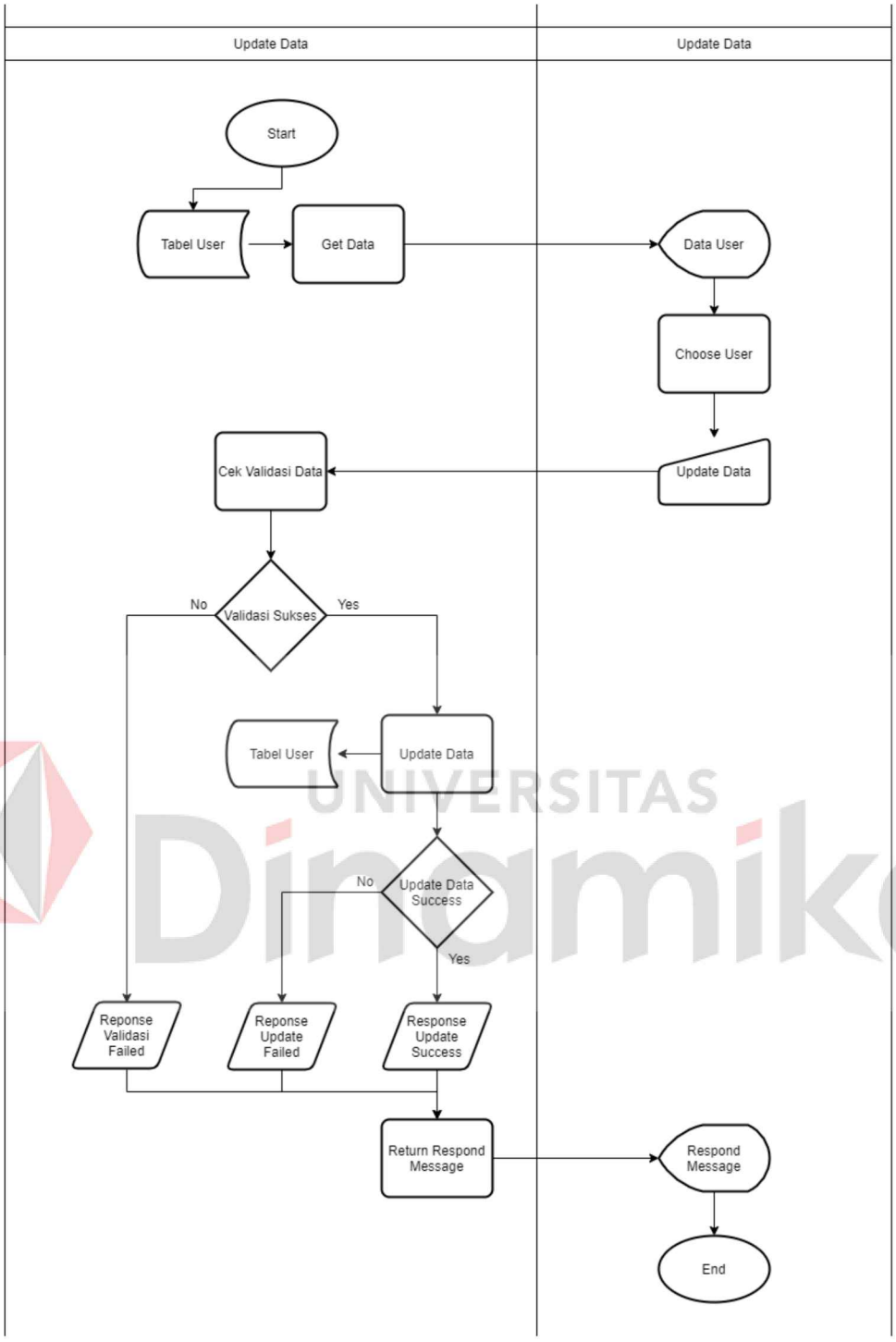
1. *Sysflow Master User*



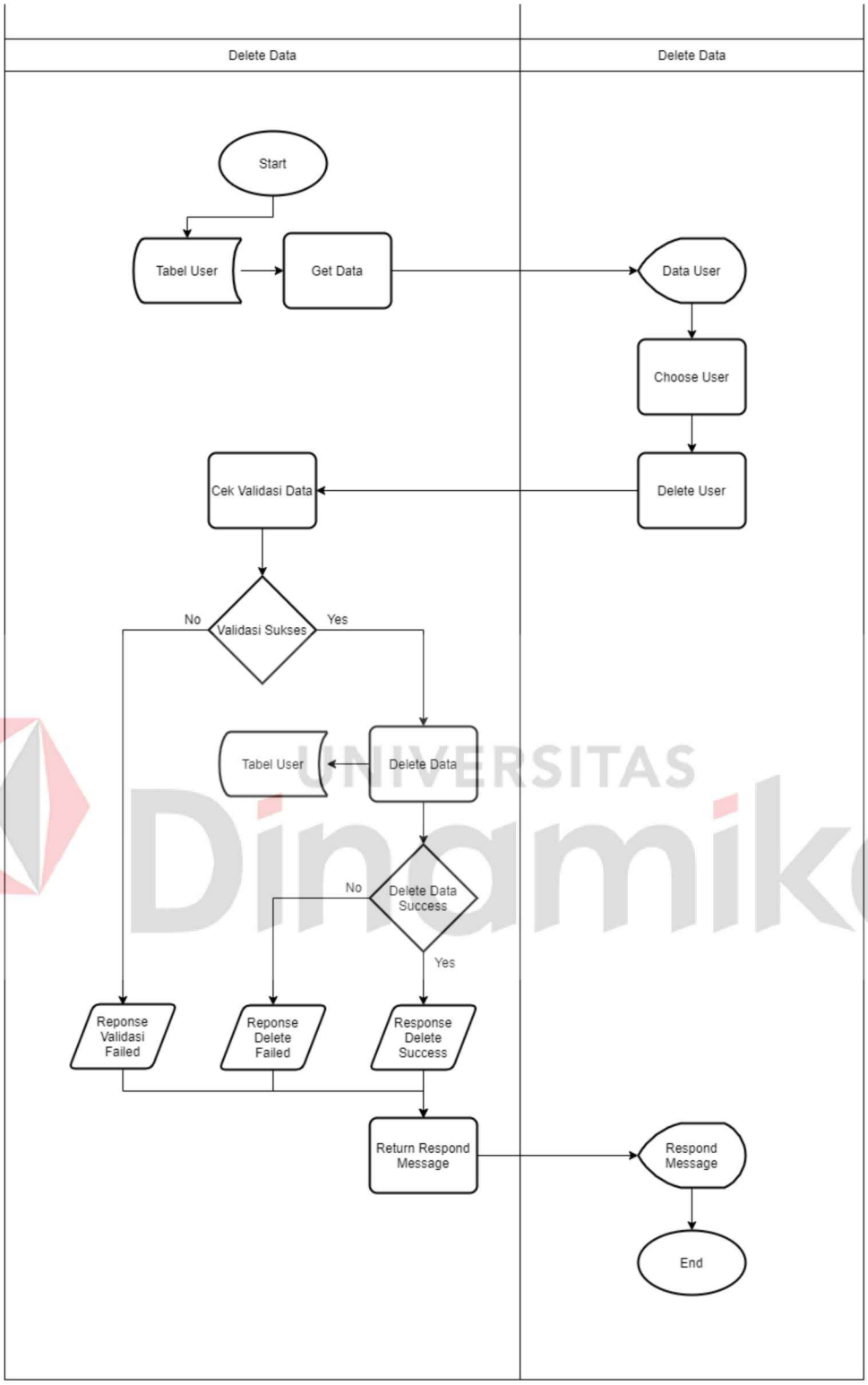
Gambar 4. 1 *Sysflow Master User Show* Data



Gambar 4. 2 *Sysflow Master User Insert* Data



Gambar 4. 3 *Sysflow Master User Update* Data



Gambar 4. 4 *Sysflow Master User Delete* Data

Pada *Sysflow User* ini terdiri dari 4 bagian, yaitu *show* data, *Insert* data, *Update* data dan *delete* data. Pada *show* data, *system* akan mengambil data dari table *user* dan nantinya akan ditampilkan di tampilan milik *user*.

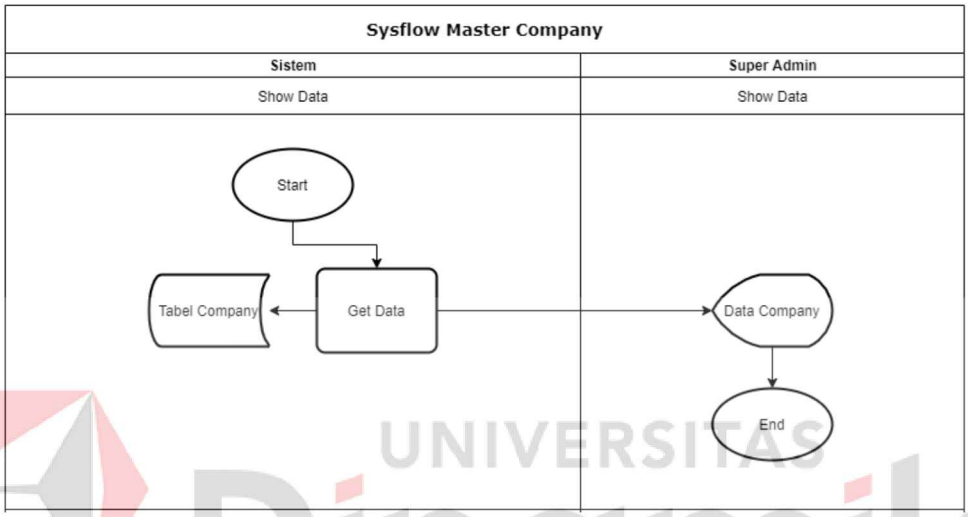
Insert Data dimulai dengan *user* memasukan data-data yang diperlukan. Data-data yang telah diinputkan dikirimkan ke *system* dan diproses. Pertama-tama data akan divalidasi, dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Insert* data ke tabel *user*. Jika proses *Insert* berhasil maka sistem akan mengembalikan pesan *Insert* data success dan *Insert* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

Update Data dimulai dengan *system* menampilkan data *user* dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data dari *user* mana yang akan dirubah. *User* merubah data-data yang diperlukan. Data-data yang telah diubah akan dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Update* data ke tabel *user*. Jika proses *Update* berhasil maka sistem akan mengembalikan pesan *Update* data success dan *Update* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

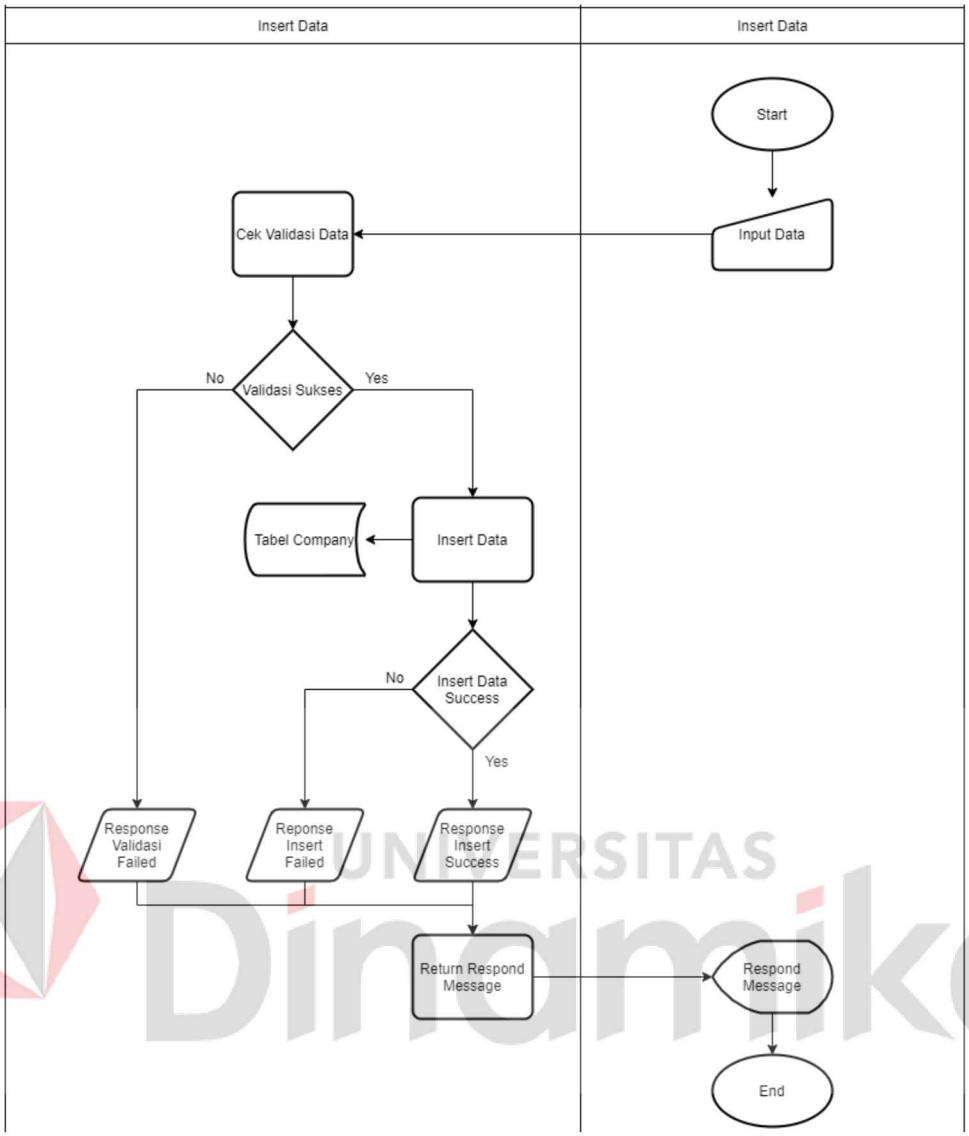
Delete Data dimulai dengan *system* menampilkan data *user* dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data dari *user* mana yang akan dirubah. *User* akan memilih menu *delete* data lalu data tersebut dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan

mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *delete* data dari tabel *user*. Jika proses *delete* berhasil maka sistem akan mengembalikan pesan *delete* data success dan *delete* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

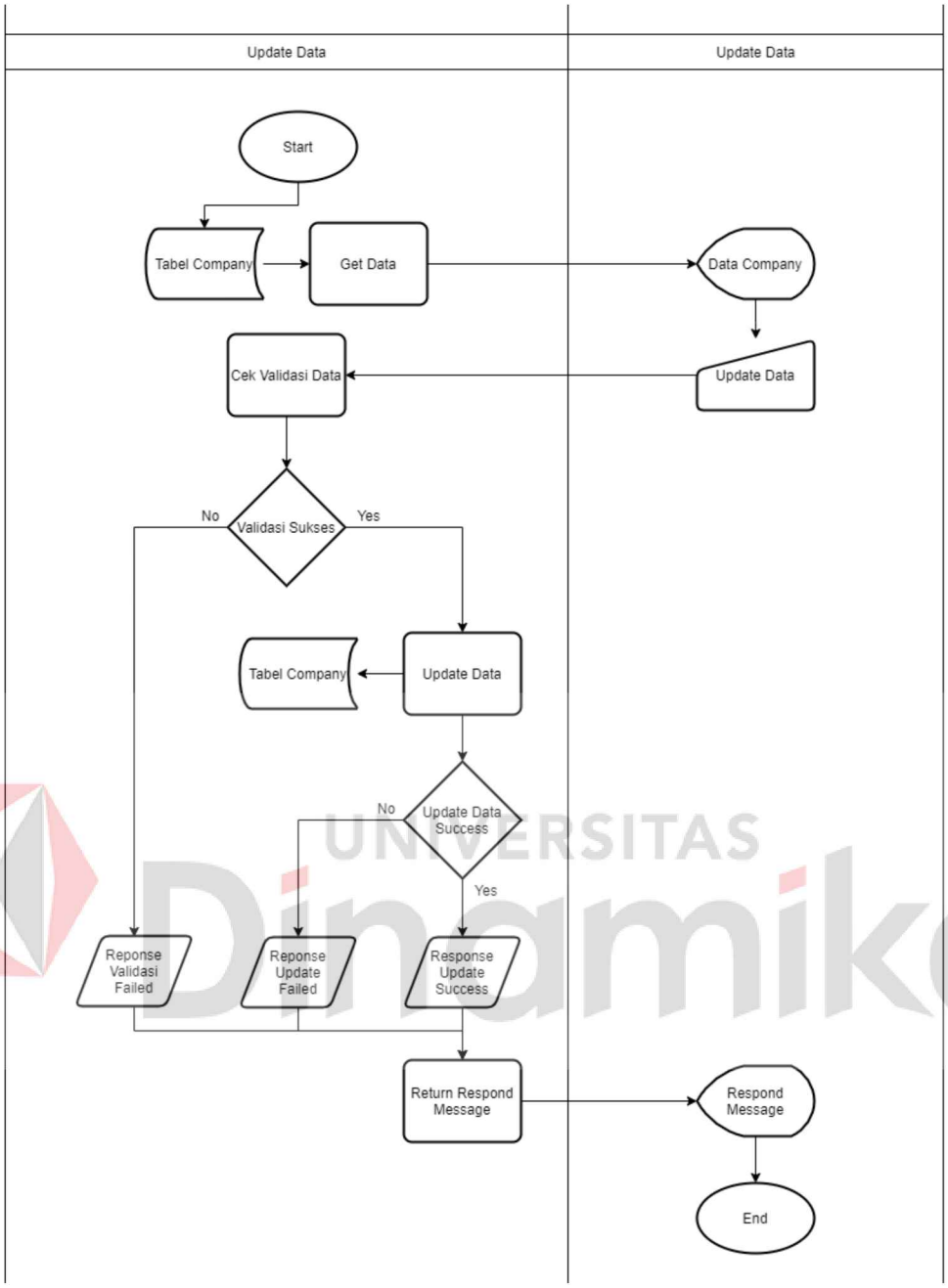
2. *Sysflow Master Company*



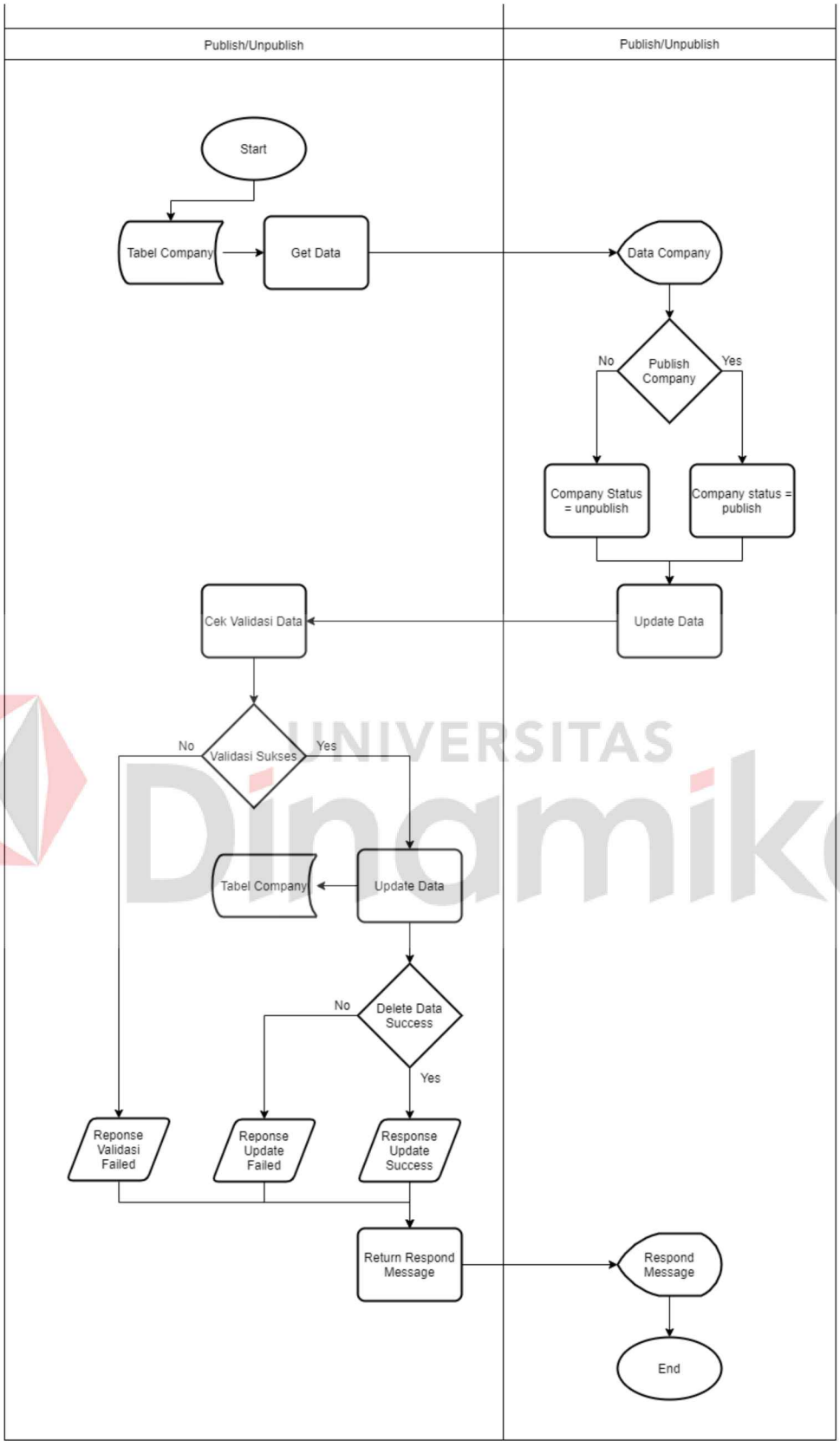
Gambar 4. 5 *Sysflow Master Company Show* Data



Gambar 4. 6 *Sysflow Master Company Insert* Data



Gambar 4. 7 *Sysflow Master Company Update* Data



Gambar 4. 8 *Sysflow Master Company Publish/ UnPublish*

Pada *Sysflow Company* ini terdiri dari 4 bagian, yaitu *show* data, *Insert* data, *Update* data dan *Publish*/*UnPublish Company*. Pada *show* data, *system* akan mengambil data dari table *Company* dan nantinya akan ditampilkan di tampilan milik *user*.

Insert Data dimulai dengan *user* memasukan data-data yang diperlukan. Data-data yang telah diinputkan dikirimkan ke *system* dan diproses. Pertama-tama data akan divalidasi, dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Insert* data ke tabel *Company*. Jika proses *Insert* berhasil maka sistem akan mengembalikan pesan *Insert* data success dan *Insert* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

Update Data dimulai dengan *system* menampilkan data *Company* dari tabel ke tampilan milik *user*. *User* dapat merubah data dari *Company*. Data-data yang telah diubah akan dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Update* data ke tabel *Company*. Jika proses *Update* berhasil maka sistem akan mengembalikan pesan *Update* data success dan *Update* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

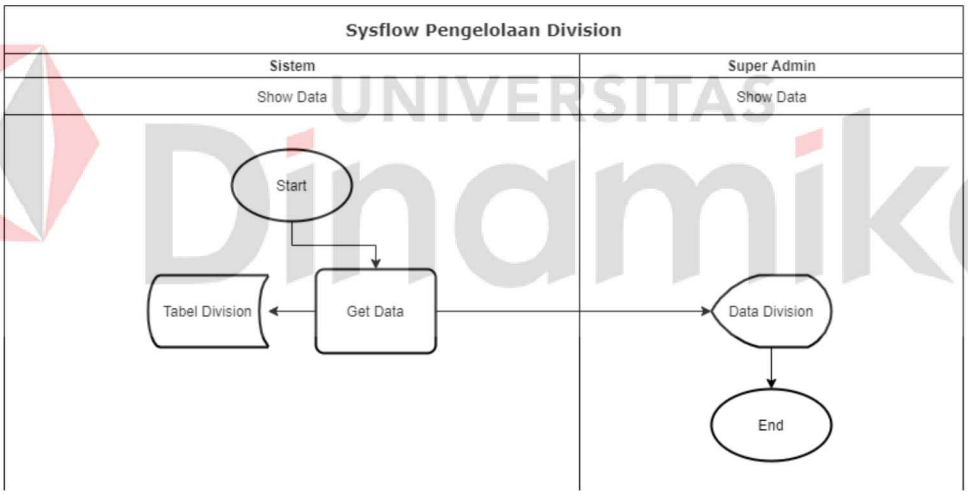
Publish atau *UnPublish Company* dimulai dengan *system* menampilkan data *Company* dari tabel ke tampilan milik *user*. *User* akan memilih menu *Publish* atau *UnPublish Company* sesuai dengan kebutuhan. Lalu data tersebut dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan

mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Update* data pada tabel *Company*. Jika proses *Update* data berhasil maka sistem akan mengembalikan pesan *Update* data success dan *Update* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

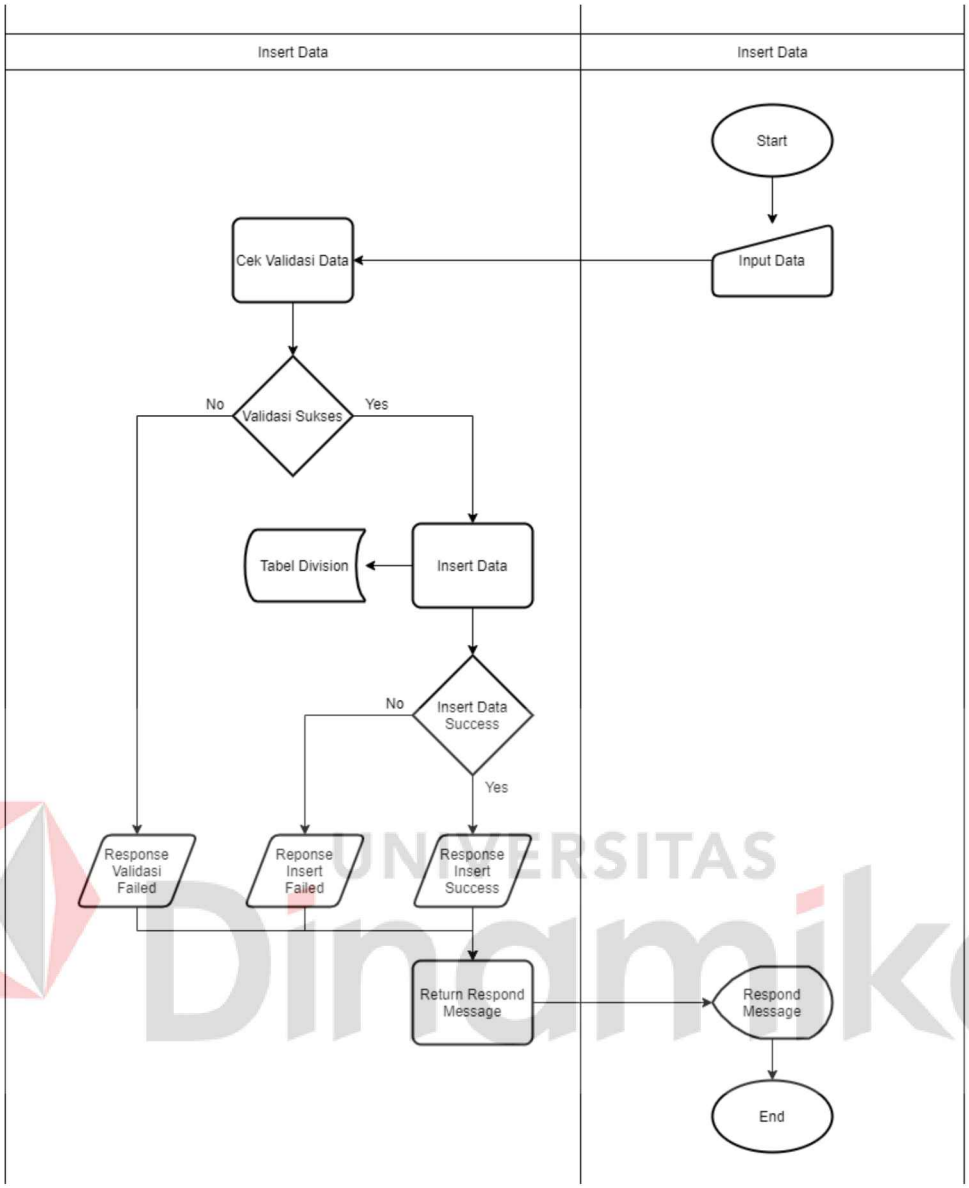
4.6.2 *Sysflow* Pengelolaan

Pada *Sysflow* pengelolaan dibagi menjadi *Sysflow* pengelolaan *Division*, *Sysflow* pengelolaan *ChatBot*, *Sysflow* pengelolaan *Chat Default* dan *Sysflow* pengelolaan FAQ. Berikut merupakan penjelasan dari *Sysflow* – *Sysflow* terkait.

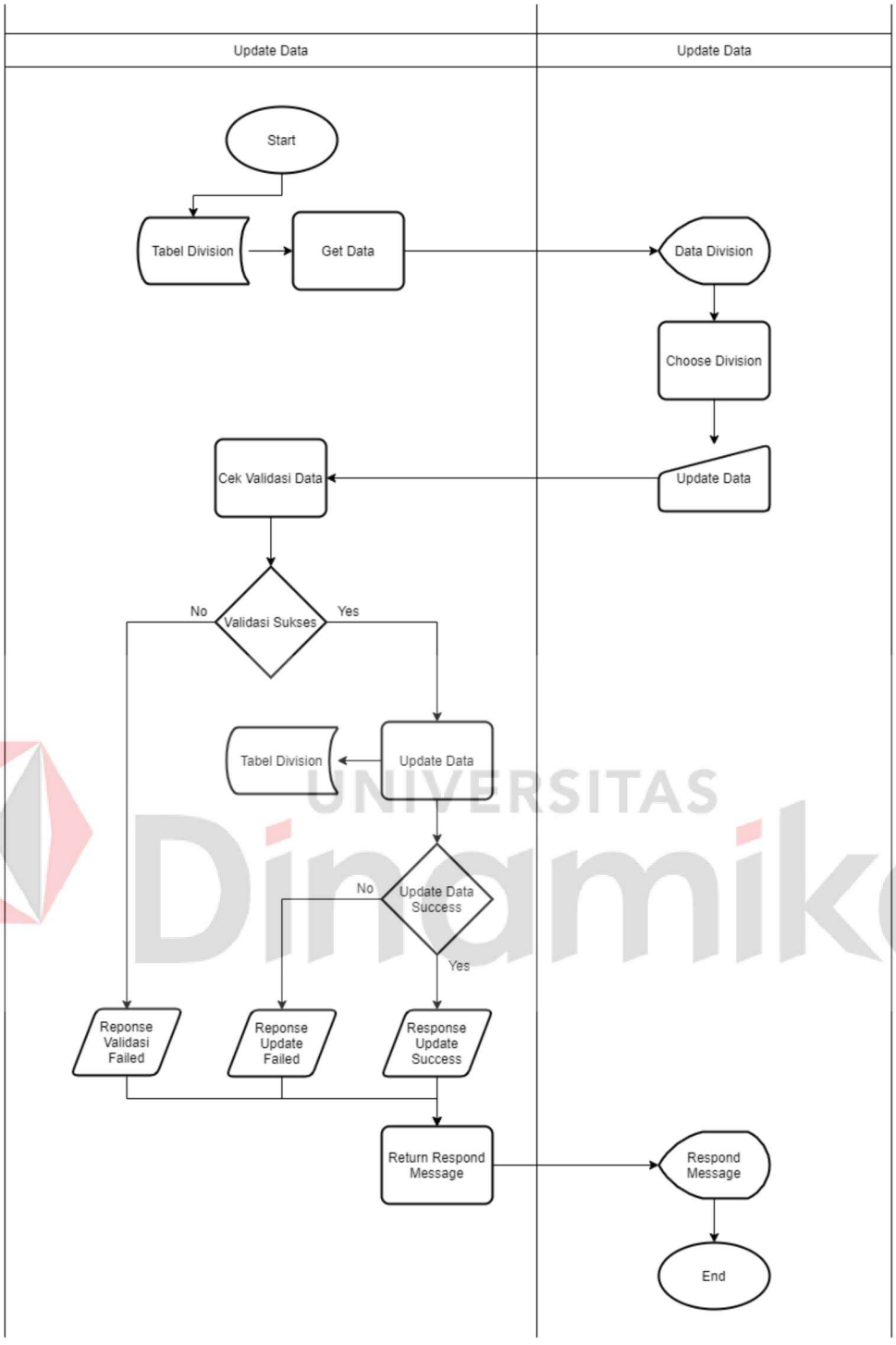
1. *Sysflow* Pengelolaan *Division*



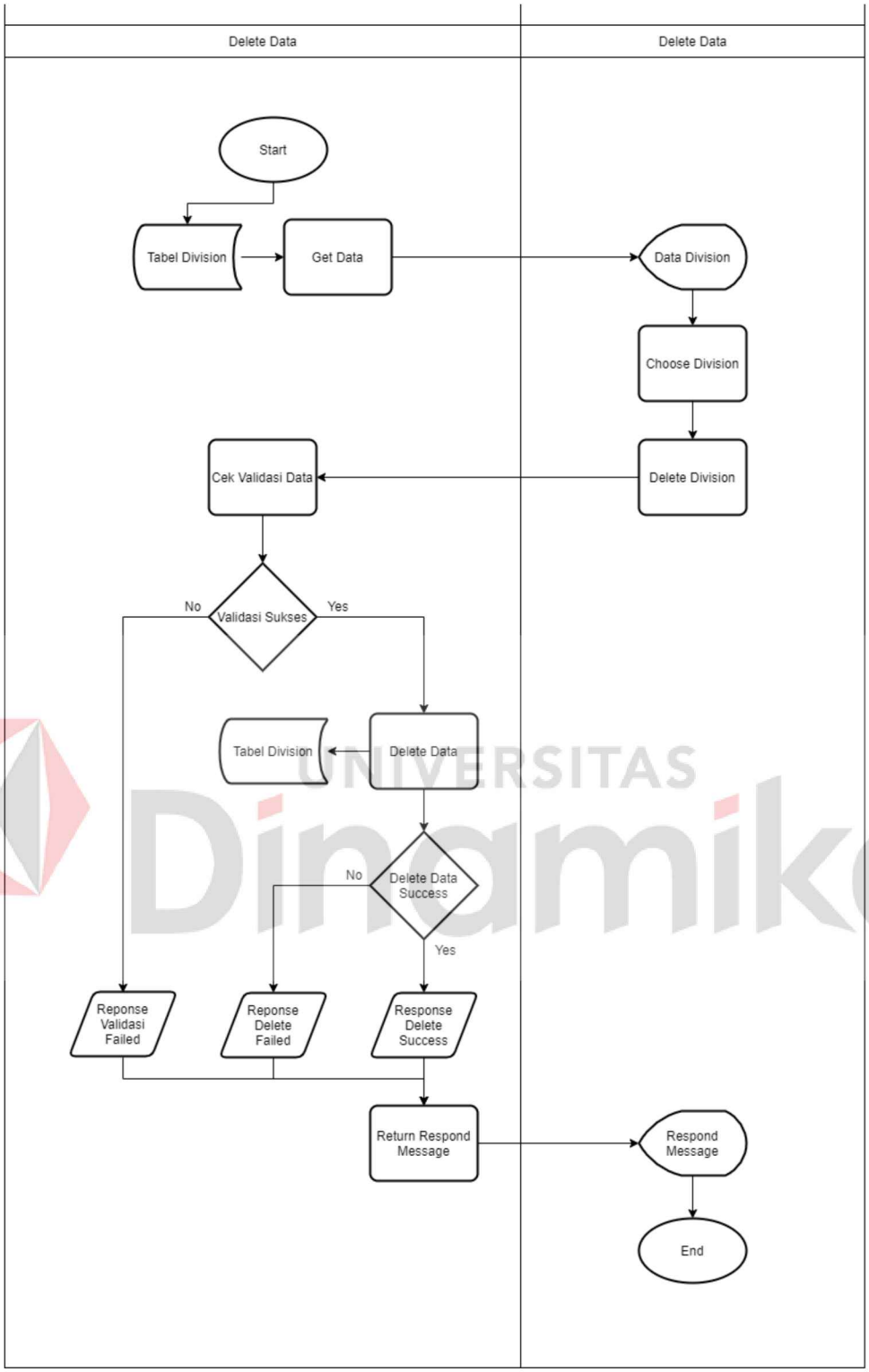
Gambar 4. 9 *Sysflow* Pengelolaan *Division Show* Data



Gambar 4. 10 *Sysflow* Pengelolaan *Division Insert* Data



Gambar 4. 11 *Sysflow* Pengelolaan *Division Update* Data



Gambar 4. 12 *Sysflow* Pengelolaan *Division Delete* Data

Pada *Sysflow Division* ini terdiri dari 4 bagian, yaitu *show* data, *Insert* data, *Update* data dan *delete* data. Pada *show* data, *system* akan mengambil data dari table *Division* dan nantinya akan ditampilkan di tampilan milik *user*.

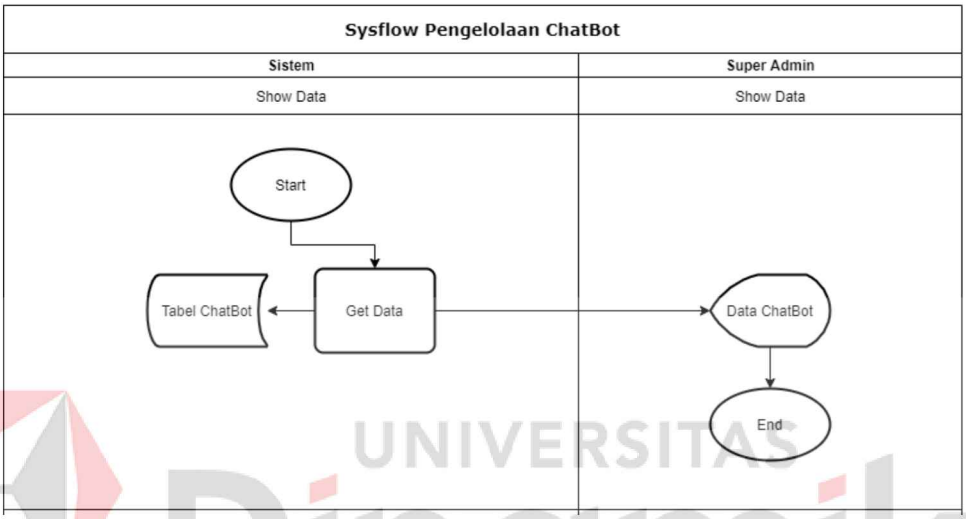
Insert Data dimulai dengan *user* memasukan data-data yang diperlukan. Data-data yang telah diinputkan dikirimkan ke *system* dan diproses. Pertama-tama data akan divalidasi, dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Insert* data ke tabel *Division*. Jika proses *Insert* berhasil maka sistem akan mengembalikan pesan *Insert* data success dan *Insert* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

Update Data dimulai dengan *system* menampilkan data *Division* dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data dari *Division* mana yang akan dirubah. *User* merubah data-data yang diperlukan. Data-data yang telah diubah akan dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Update* data ke tabel *Division*. Jika proses *Update* berhasil maka sistem akan mengembalikan pesan *Update* data success dan *Update* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

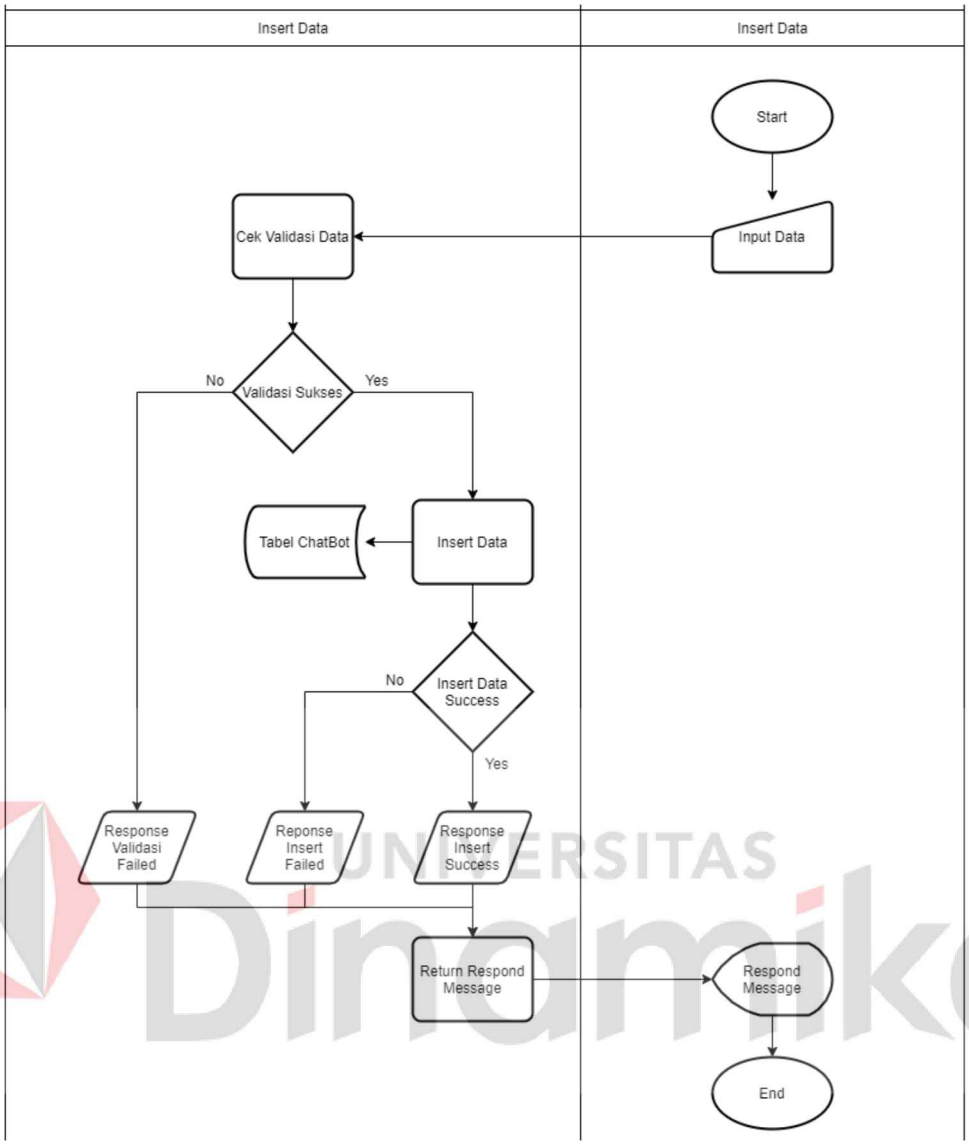
Delete Data dimulai dengan *system* menampilkan data *Division* dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data dari *Division* mana yang akan dirubah. *User* akan memilih menu *delete* data lalu data tersebut dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal

sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *delete* data dari tabel *Division*. Jika proses *delete* berhasil maka sistem akan mengembalikan pesan *delete* data success dan *delete* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

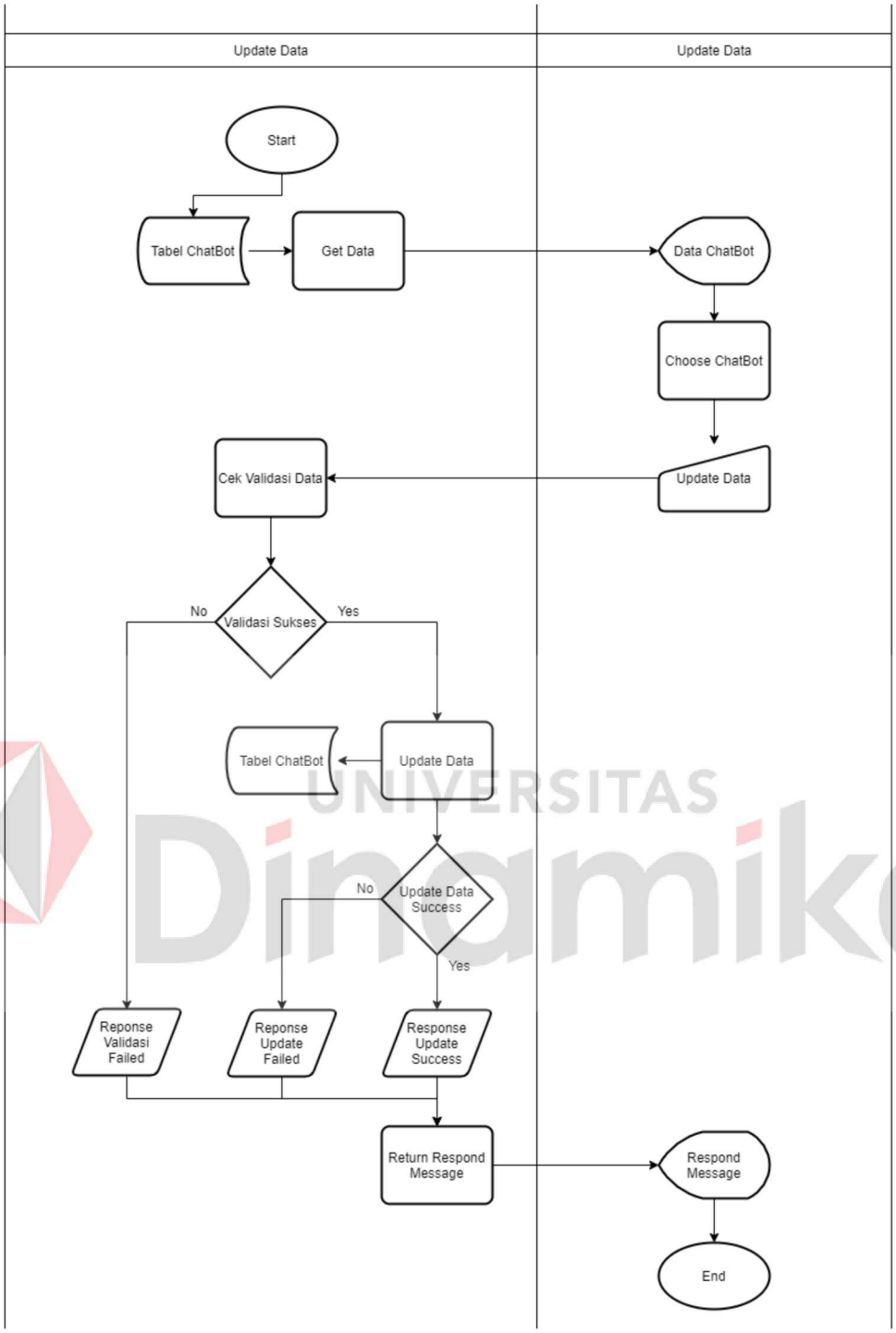
2. *Sysflow* Pengelolaan *ChatBot*



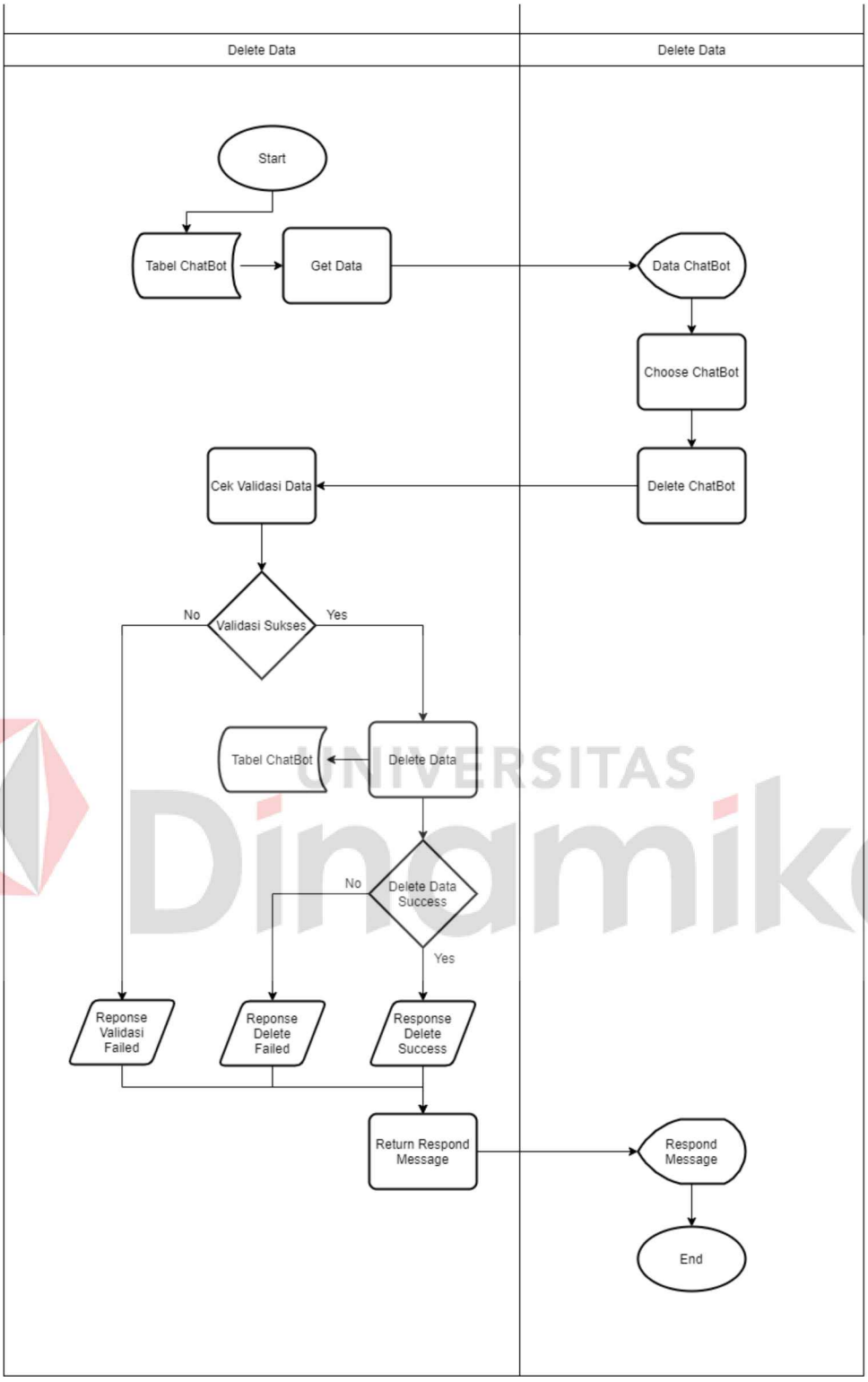
Gambar 4. 13 *Sysflow* Pengelolaan *ChatBot Show* Data



Gambar 4. 14 *Sysflow* Pengelolaan *ChatBot Insert* Data



Gambar 4. 15 *Sysflow* Pengelolaan *ChatBot Update* Data



Gambar 4. 16 *Sysflow* Pengelolaan *ChatBot Delete* Data

Pada *Sysflow ChatBot* ini terdiri dari 4 bagian, yaitu *show* data, *Insert* data, *Update* data dan *delete* data. Pada *show* data, *system* akan mengambil data dari table *ChatBot* dan nantinya akan ditampilkan di tampilan milik *user*.

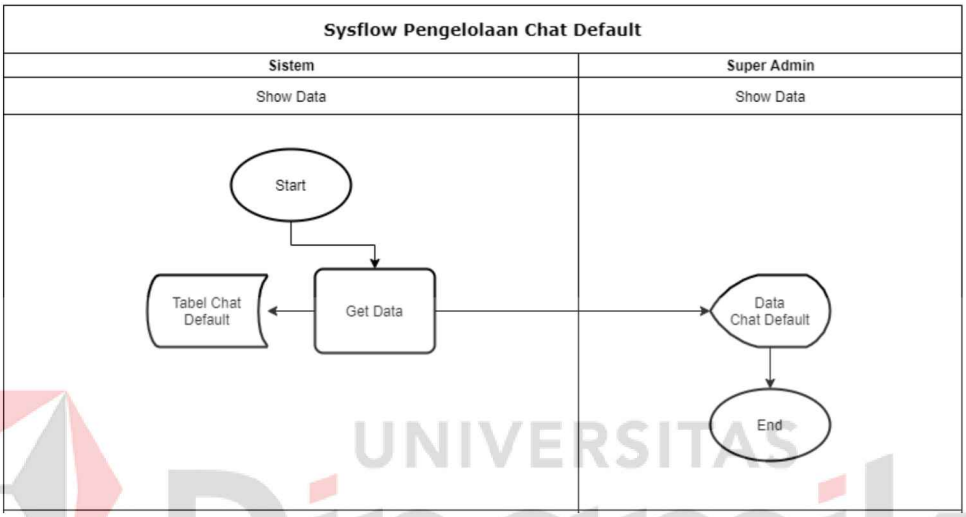
Insert Data dimulai dengan *user* memasukan data-data yang diperlukan. Data-data yang telah diinputkan dikirimkan ke *system* dan diproses. Pertama-tama data akan divalidasi, dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Insert* data ke tabel *ChatBot*. Jika proses *Insert* berhasil maka sistem akan mengembalikan pesan *Insert* data success dan *Insert* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

Update Data dimulai dengan *system* menampilkan data *ChatBot* dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data dari *ChatBot* mana yang akan dirubah. *User* merubah data-data yang diperlukan. Data-data yang telah diubah akan dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Update* data ke tabel *ChatBot*. Jika proses *Update* berhasil maka sistem akan mengembalikan pesan *Update* data success dan *Update* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

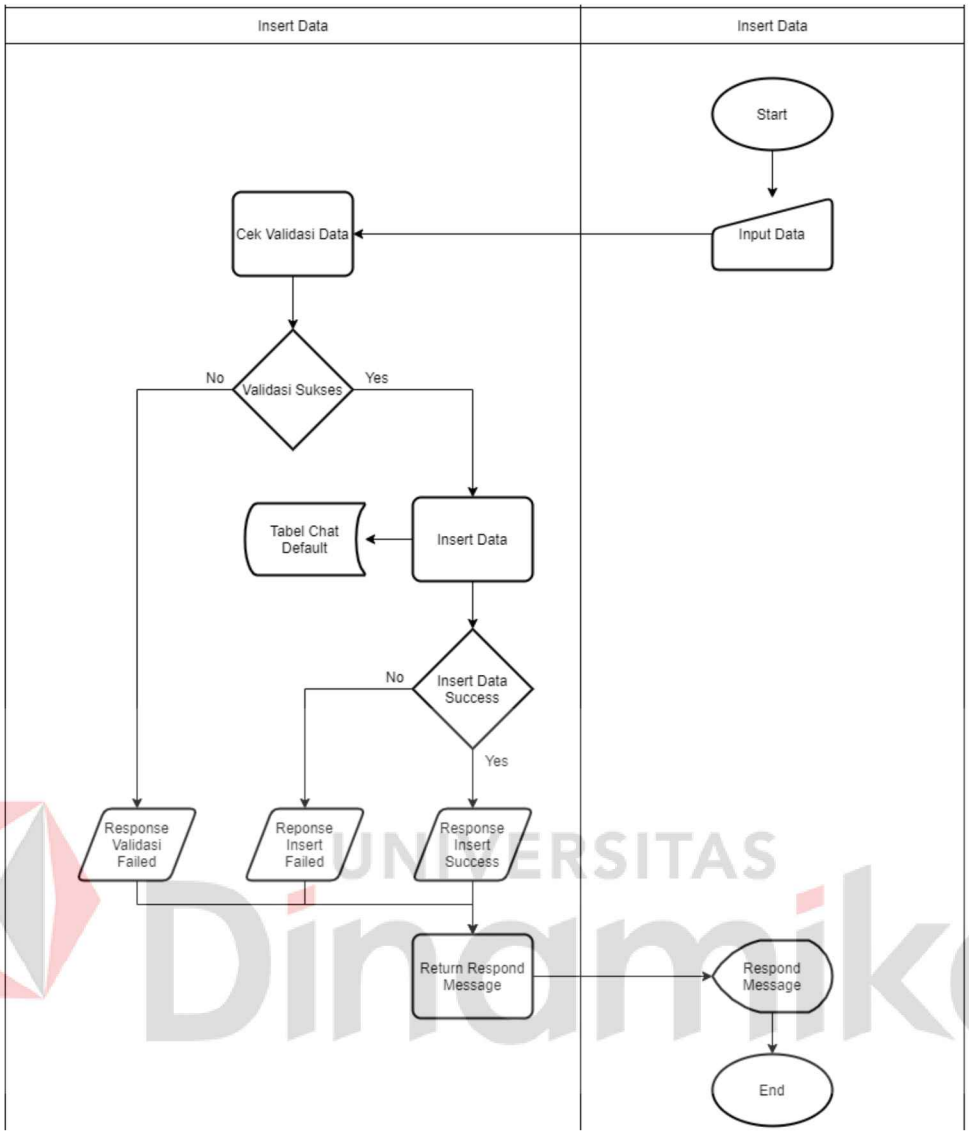
Delete Data dimulai dengan *system* menampilkan data *ChatBot* dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data dari *ChatBot* mana yang akan dirubah. *User* akan memilih menu *delete* data lalu data tersebut dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan

mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *delete* data dari tabel *Division*. Jika proses *delete* berhasil maka sistem akan mengembalikan pesan *delete* data success dan *delete* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

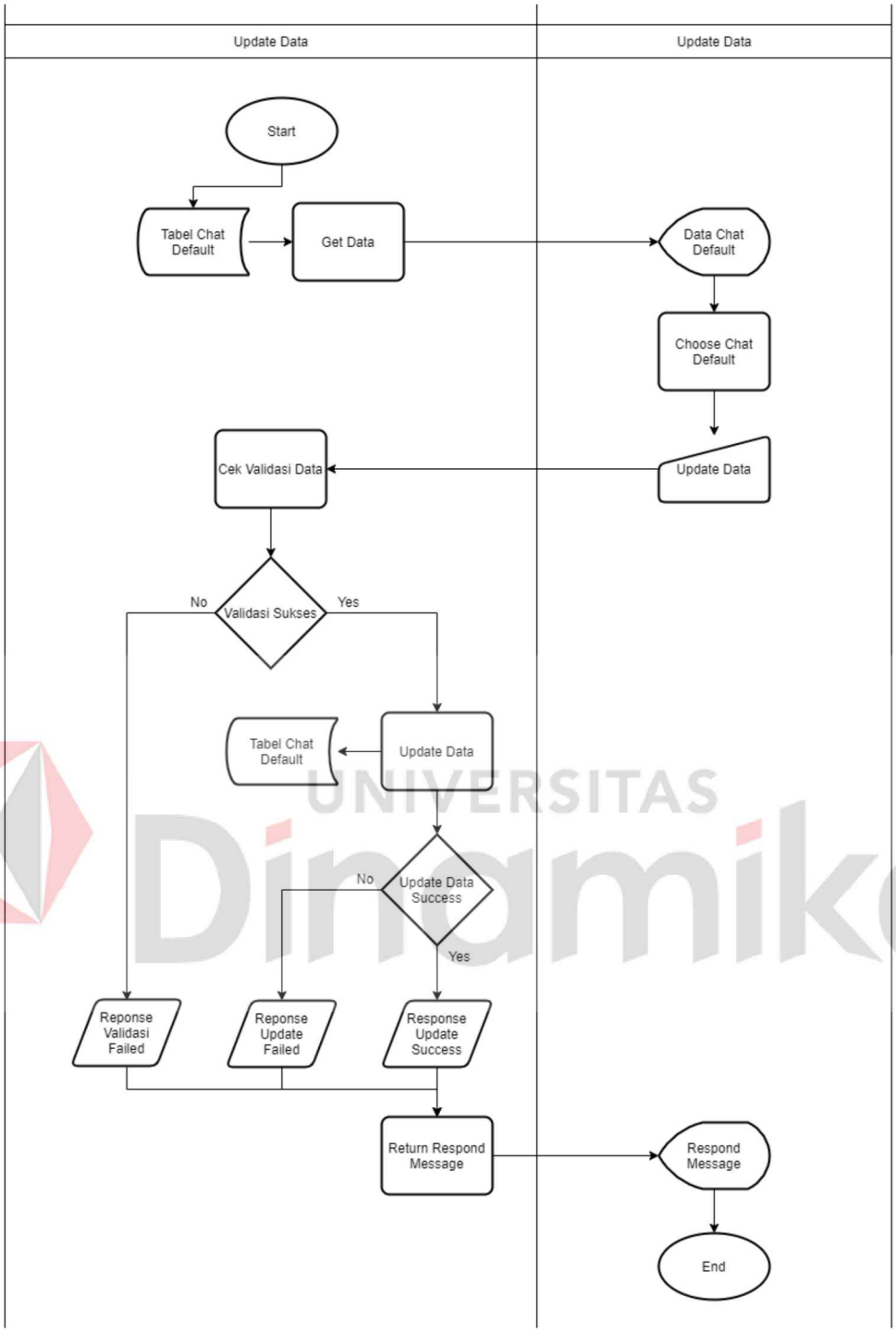
3. *Sysflow* Pengelolaan *Chat Default*



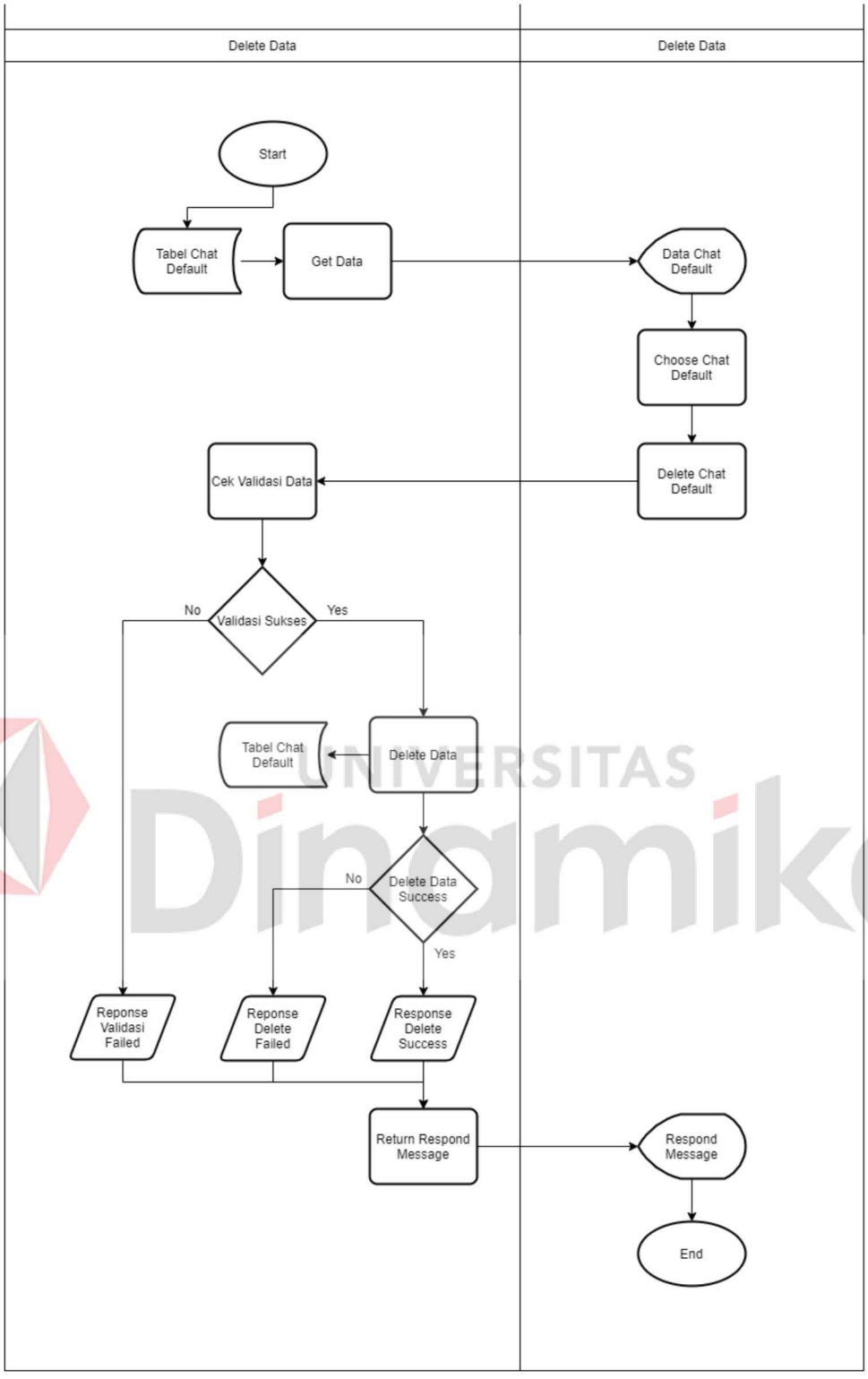
Gambar 4. 17 *Sysflow* Pengelolaan *Chat Default Show* Data



Gambar 4. 18 *Sysflow* Pengelolaan *Chat Default Insert* Data



Gambar 4. 19 *Sysflow* Pengelolaan *Chat Default Update* Data



Gambar 4. 20 *Sysflow* Pengelolaan *Chat Default Delete* Data

Pada *Sysflow Chat Default* ini terdiri dari 4 bagian, yaitu *show* data, *Insert* data, *Update* data dan *delete* data. Pada *show* data, *system* akan mengambil data dari table *Chat Default* dan nantinya akan ditampilkan di tampilan milik *user*.

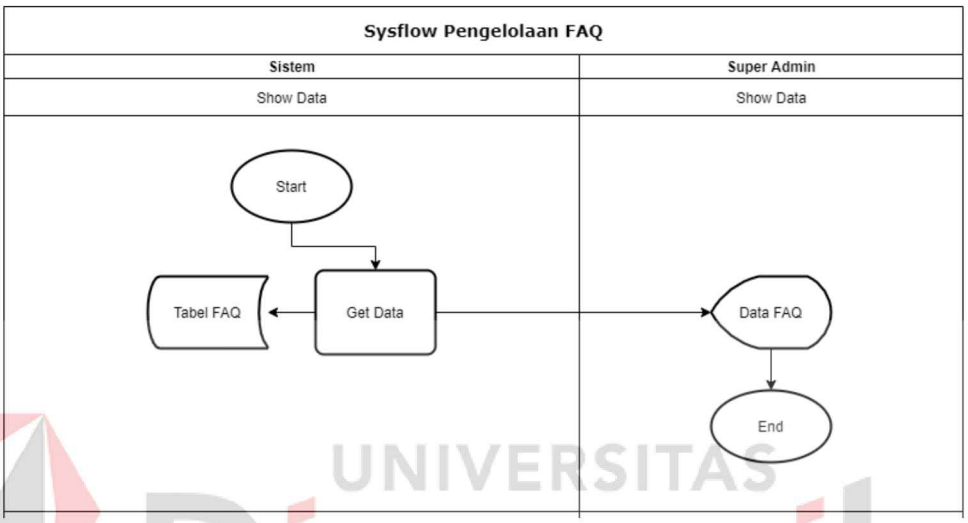
Insert Data dimulai dengan *user* memasukan data-data yang diperlukan. Data-data yang telah diinputkan dikirimkan ke *system* dan diproses. Pertama-tama data akan divalidasi, dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Insert* data ke tabel *Chat Default*. Jika proses *Insert* berhasil maka sistem akan mengembalikan pesan *Insert* data success dan *Insert* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

Update Data dimulai dengan *system* menampilkan data *Chat Default* dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data dari *Chat Default* mana yang akan dirubah. *User* merubah data-data yang diperlukan. Data-data yang telah diubah akan dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Update* data ke tabel *Chat Default*. Jika proses *Update* berhasil maka sistem akan mengembalikan pesan *Update* data success dan *Update* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

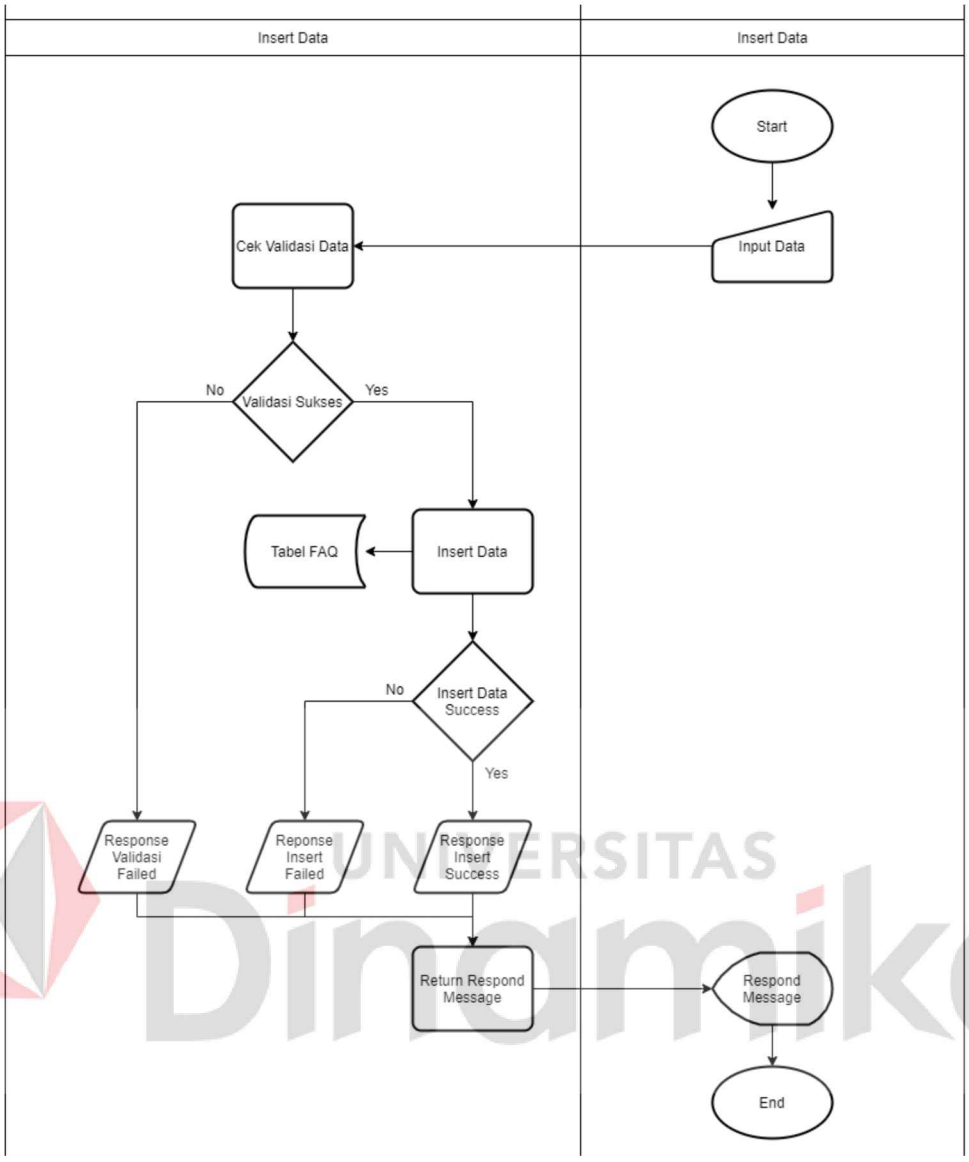
Delete Data dimulai dengan *system* menampilkan data *Chat Default* dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data mana yang akan dihapus. *User* akan memilih menu *delete* data lalu data tersebut dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan

mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *delete* data dari tabel *Chat Default*. Jika proses *delete* berhasil maka sistem akan mengembalikan pesan *delete* data success dan *delete* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

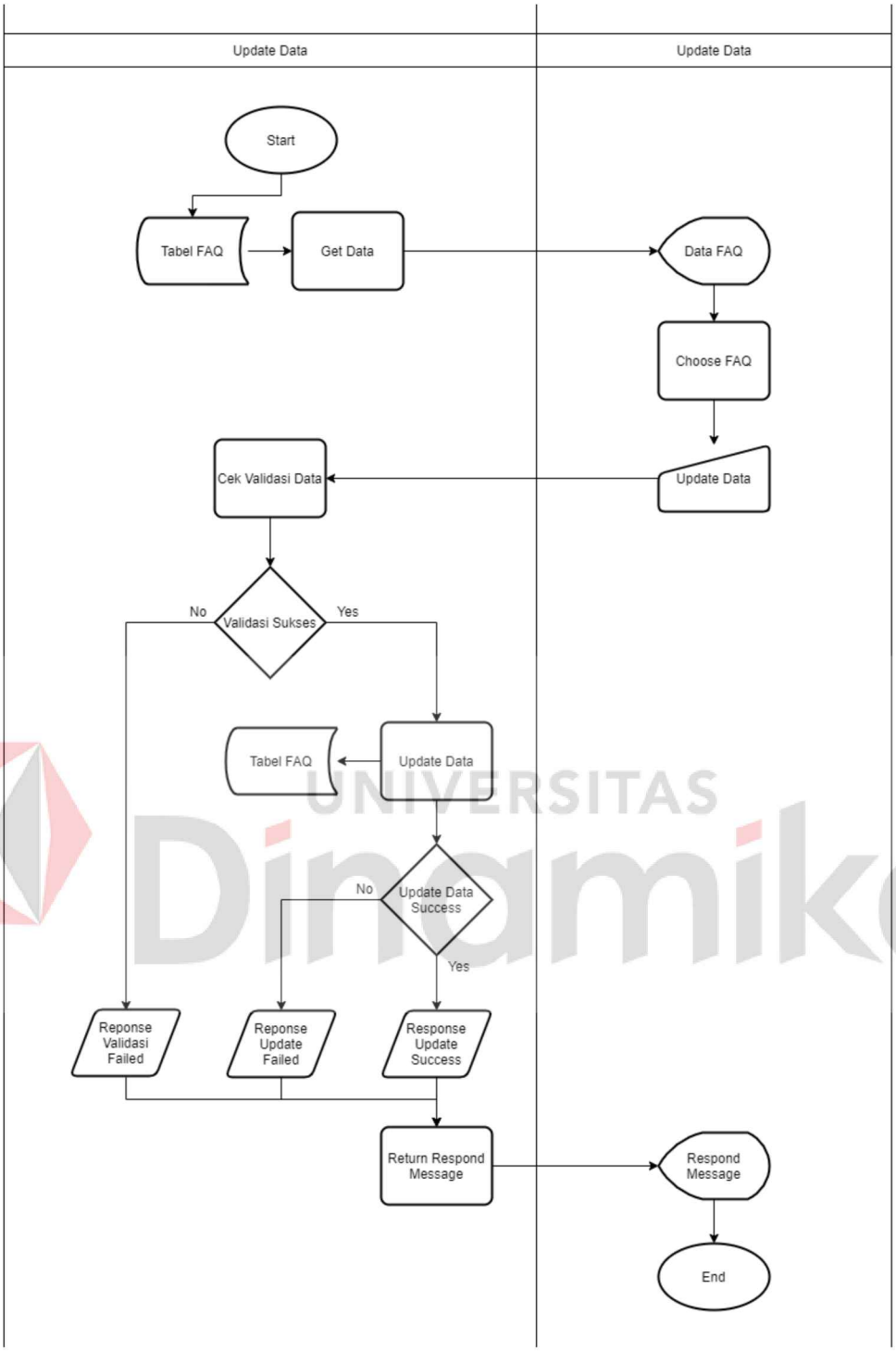
4. *Sysflow* Pengelolaan FAQ



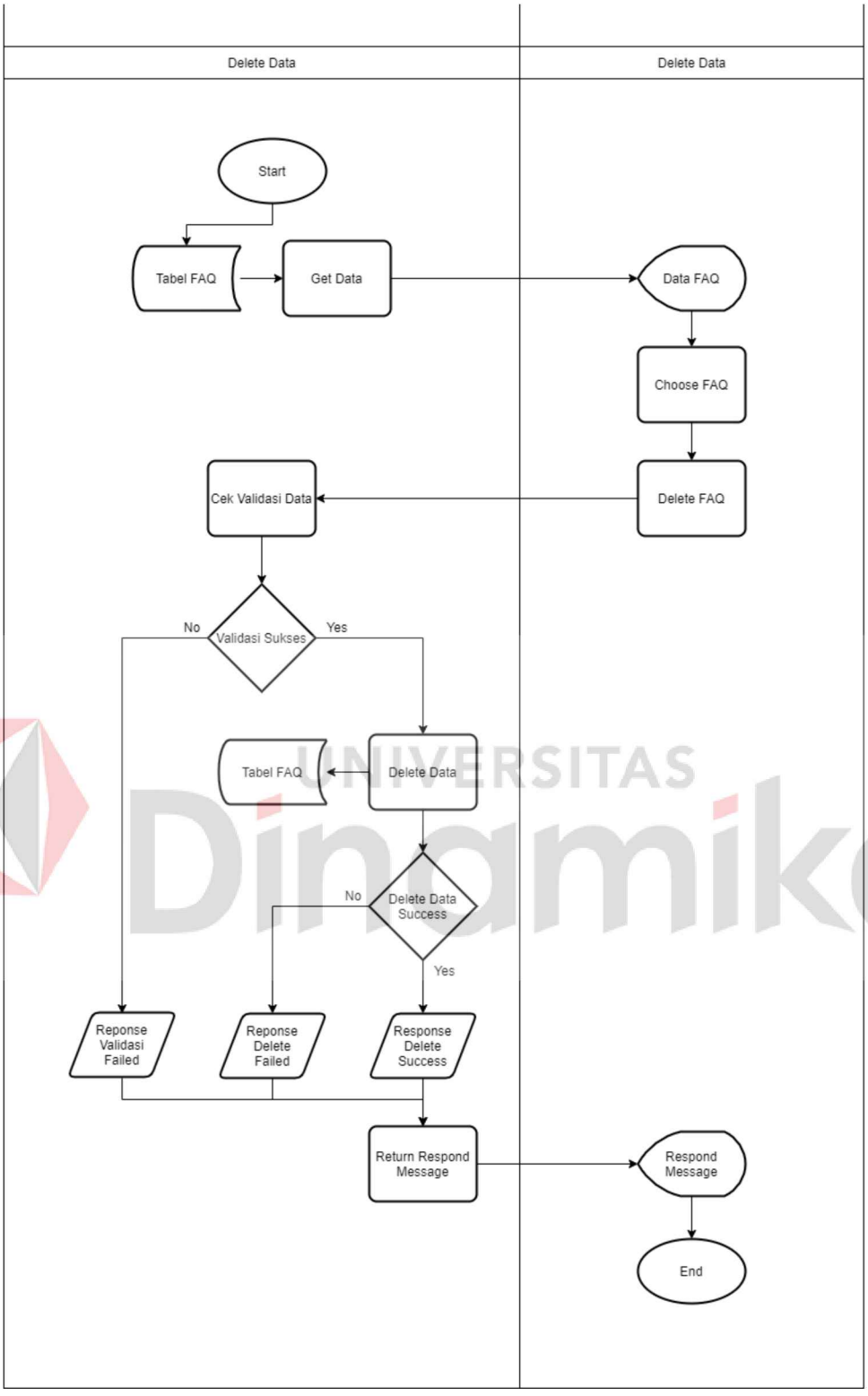
Gambar 4. 21 *Sysflow* Pengelolaan FAQ *Show* Data



Gambar 4. 22 *Sysflow* Pengelolaan FAQ *Insert* Data



Gambar 4. 23 *Sysflow* Pengelolaan FAQ *Update* Data



Gambar 4. 24 *Sysflow* Pengelolaan FAQ *Delete* Data

Pada *Sysflow* FAQ ini terdiri dari 4 bagian, yaitu *show* data, *Insert* data, *Update* data dan *delete* data. Pada *show* data, *system* akan mengambil data dari table faq dan nantinya akan ditampilkan di tampilan milik *user*.

Insert Data dimulai dengan *user* memasukan data-data yang diperlukan. Data-data yang telah diinputkan dikirimkan ke *system* dan diproses. Pertama-tama data akan divalidasi, dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Insert* data ke tabel faq. Jika proses *Insert* berhasil maka sistem akan mengembalikan pesan *Insert* data success dan *Insert* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

Update Data dimulai dengan *system* menampilkan data faq dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data dari faq mana yang akan dirubah. *User* merubah data-data yang diperlukan. Data-data yang telah diubah akan dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *Update* data ke tabel faq. Jika proses *Update* berhasil maka sistem akan mengembalikan pesan *Update* data success dan *Update* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

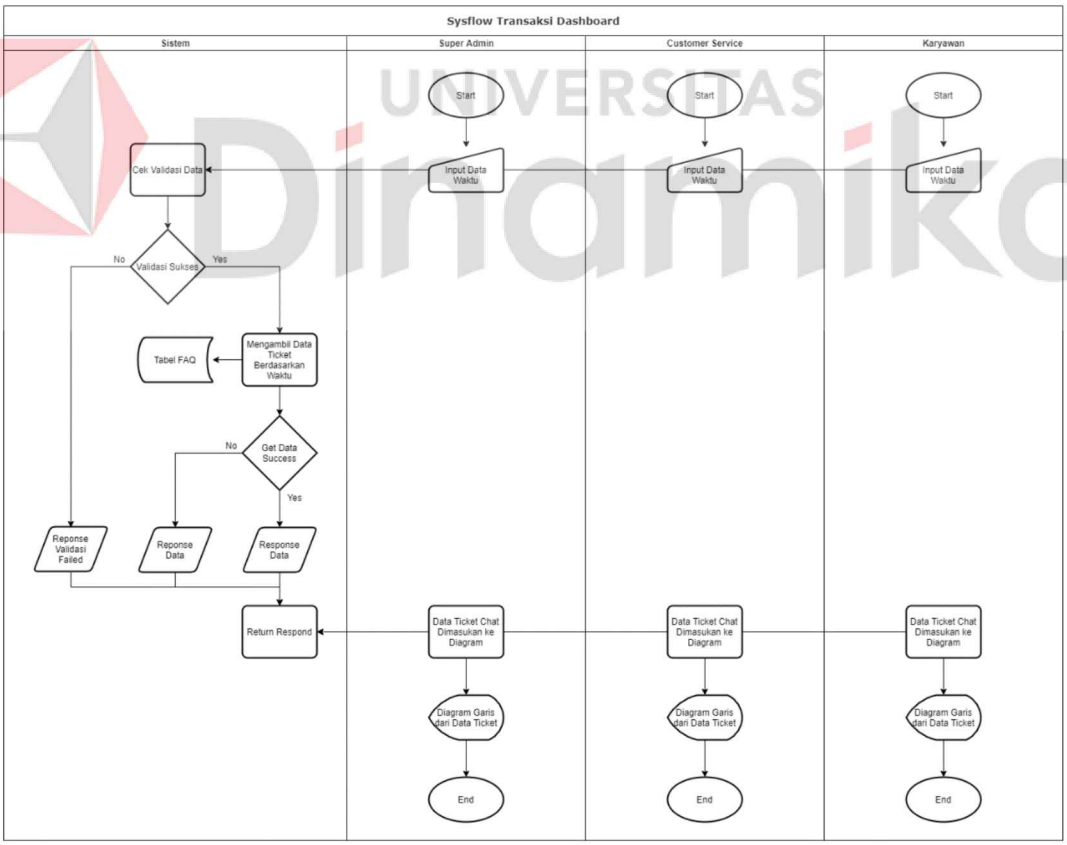
Delete Data dimulai dengan *system* menampilkan data faq dari tabel ke tampilan milik *user*. Selanjutnya *user* akan memilih data mana yang akan dihapus. *User* akan memilih menu *delete* data lalu data tersebut dikirimkan ke *system* dan diproses. Data akan divalidasi oleh *system* dan jika gagal sistem akan

mengembalikan message ke *user* berupa pesan validasi gagal, dan jika berhasil maka akan dilanjutkan ke *delete* data dari tabel faq. Jika proses *delete* berhasil maka sistem akan mengembalikan pesan *delete* data success dan *delete* data *failed* jika gagal. Pesan ini nantinya akan diterima oleh *user* dan ditampilkan.

4.6.3 *Sysflow* Transaksi

Sysflow Transaksi sendiri dibagi menjadi *Sysflow* transaksi *Dashboard* dan juga *Sysflow* transaksi *Ticket Chat*. Berikut merupakan penjelasan dari *Sysflow* transaksi *Dashboard* dan juga *Sysflow* transaksi *Ticket Chat*.

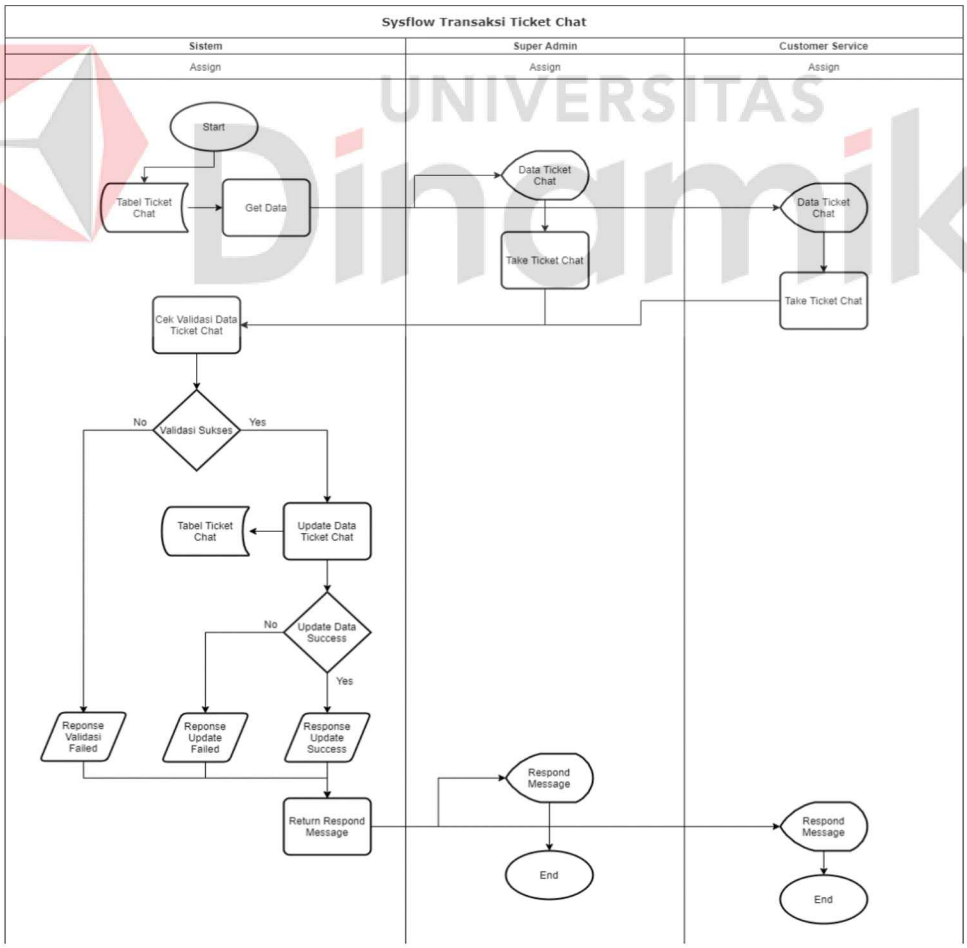
1. *Sysflow* Transaksi Dahsboard



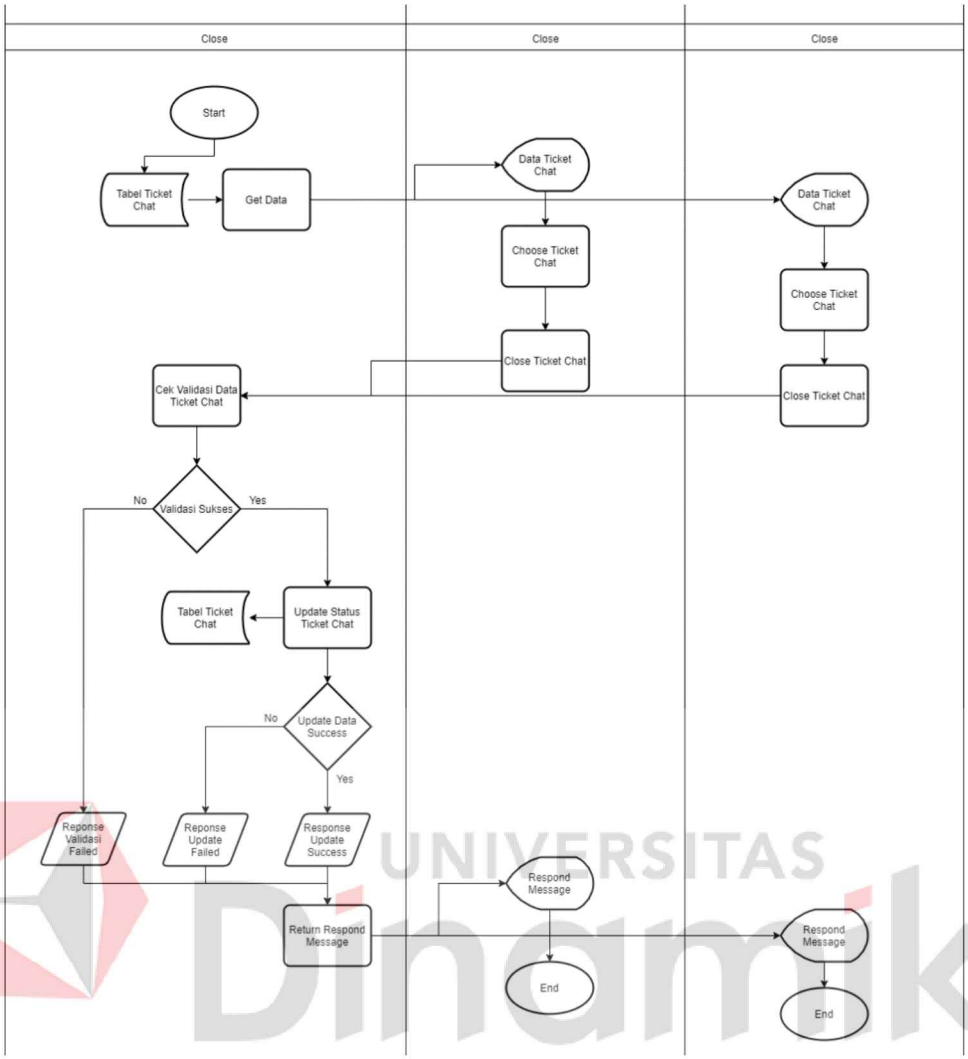
Gambar 4. 25 *Sysflow* Transaksi *Dashboard*

Pada *Sysflow* Transaksi *Dashboard* ini terdapat 3 *user* yaitu Super Admin, *Customer Service*, dan Karyawan. Ketiga *user* tersebut memiliki fungsi yang sama di dalam *Sysflow* ini. Proses dimulai dari *user* melakukan *input* data berupa rentan waktu. Setelah itu data akan dikirimkan ke *system* dan diproses. Data akan dilakukan pengecekan validasi terlebih dahulu, dan bila sukses akan dilanjutkan ke proses pengambil data pada tabel, dan bila gagal *system* akan mengembalikan sebuah pesan validasi gagal ke *user*. Setelah data diambil sistem akan mengirim data tersebut ke *user*. Pada tampilan *user* nantinya data-data yang diterima akan diolah menjadi sebuah bentuk diagram yanga dapat dibaca oleh *user*.

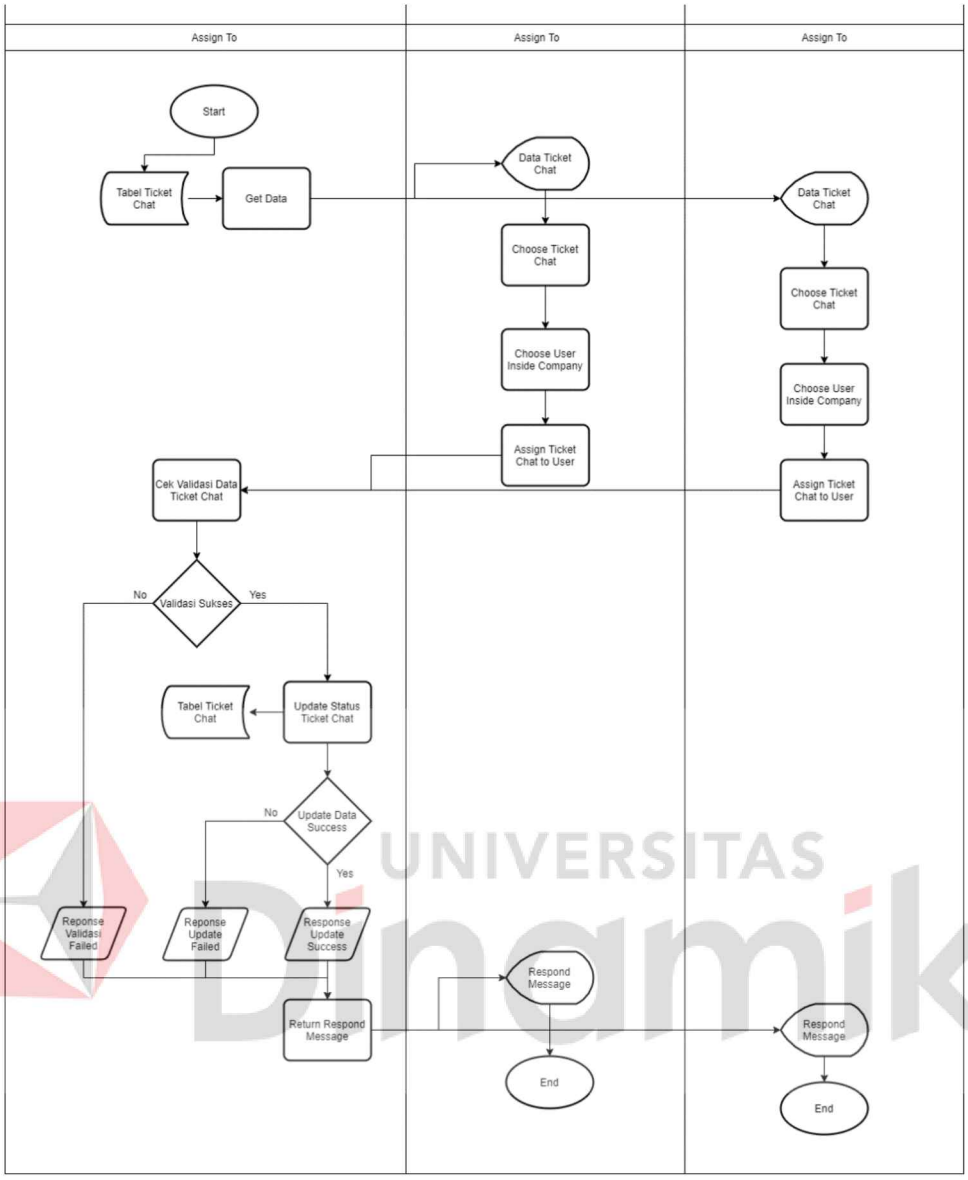
2. *Sysflow* Transaksi *Ticket Chat*



Gambar 4. 26 *Sysflow* Transaksi *Ticket Chat*



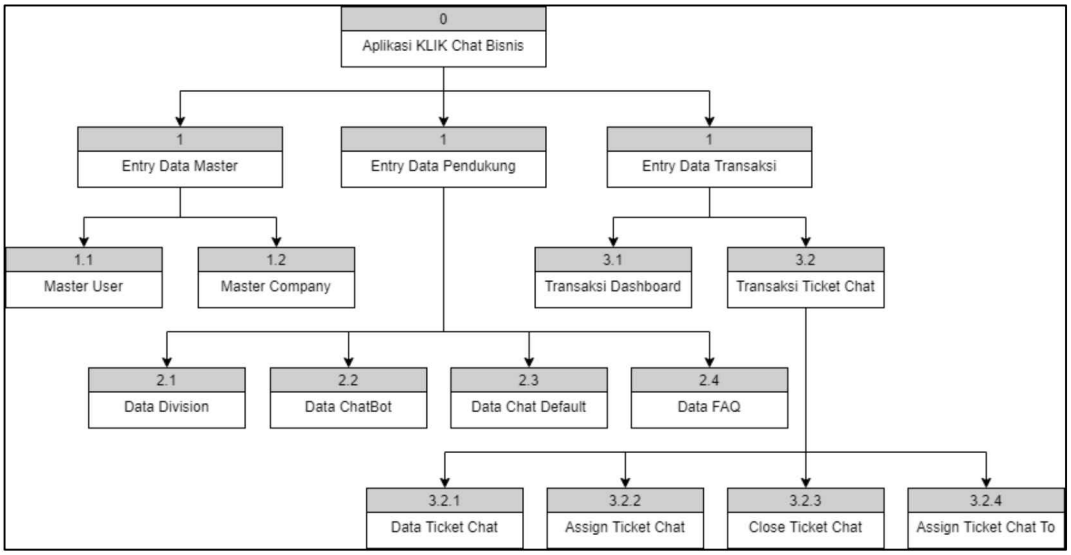
Gambar 4. 27 *Sysflow* Transaksi *Ticket Chat Close*



Gambar 4. 28 *Sysflow* Transaksi *Ticket Chat Assign To*

Pada *Sysflow* Transaksi *Ticket Chat* ini terdiri dari 3 bagian, yaitu assign, close, dan assign to pada *Ticket Chat* itu sendiri. Ketiga fungsi ini sebenarnya memiliki fungsi yang sama. Assign berarti *user* pada *Company* tersebut dapat mengambil alih *Ticket Chat* tersebut. Close berarti mengakhiri pengambil alihan *Ticket Chat* tersebut. Sedangkan assign to berfungsi untuk memindah alih kepengurusan dari ticket tersebut.

4.7 Diagram Jenjang

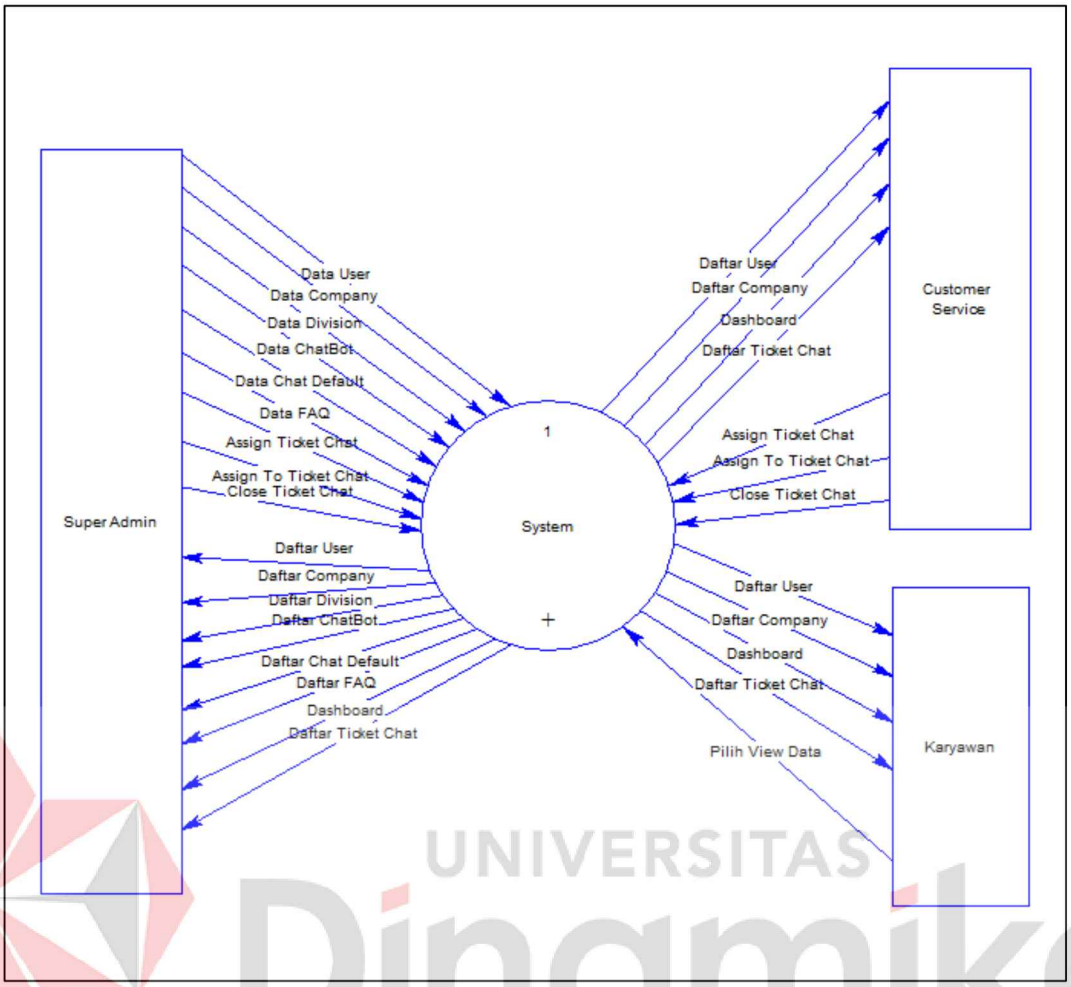


Gambar 4. 29 Diagram Jenjang

Gambar diatas merupakan gambar diagram jenjang yang telah dibuat. Terdapat beberapa level entitas menurut gambar tersebut. Gambar diagram ini nantinya berfungsi untuk pembentukan *Context Diagram*, dan DFD.

4.8 *Context Diagram*

Gambar 4.30 merupakan gambar *Context Diagram* yang merupakan penggambaran proses utama dari sistem dan juga *user* yang ada.



Gambar 4. 30 *Context Diagram*

4.9 *Data Flow Diagram* (DFD)

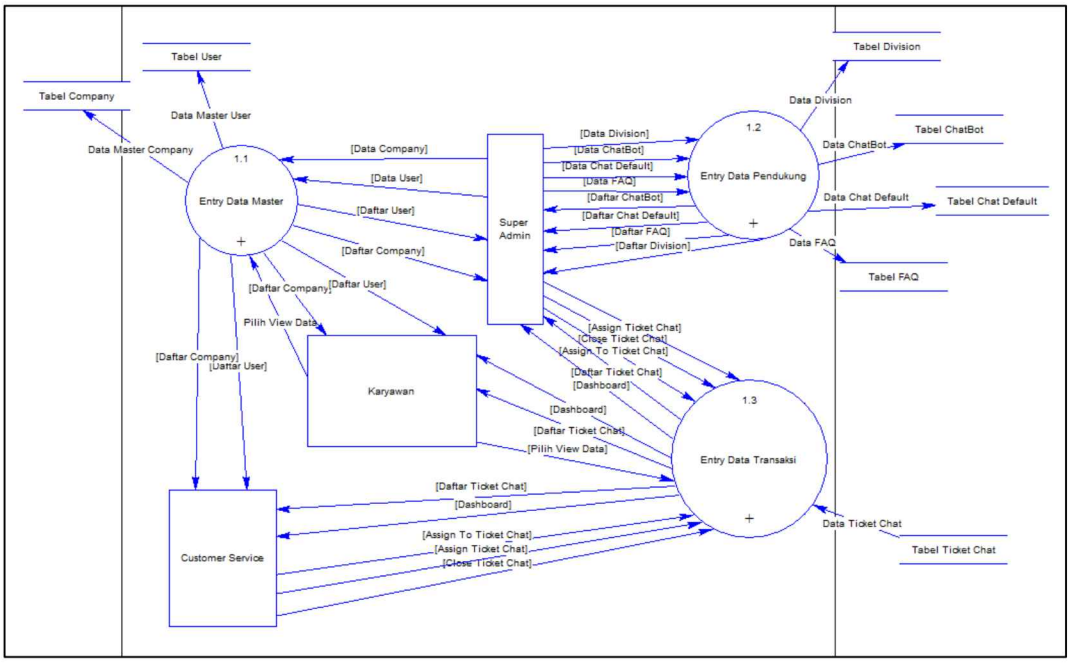
Data Flow Diagram (DFD) Dibagi menjadi beberapa level yaitu level 0, level 1 dan level 2. Berikut merupakan isi DFD dari setiap level tersebut.

1. Level 0

DFD Level 0 sendiri merupakan penjabaran proses *system* dari *Context Diagram*. Berikut merupakan daftar DFD level 0.

1.1 *System*

Berikut merupakan gambar dari proses *system* pada DFD level 0.



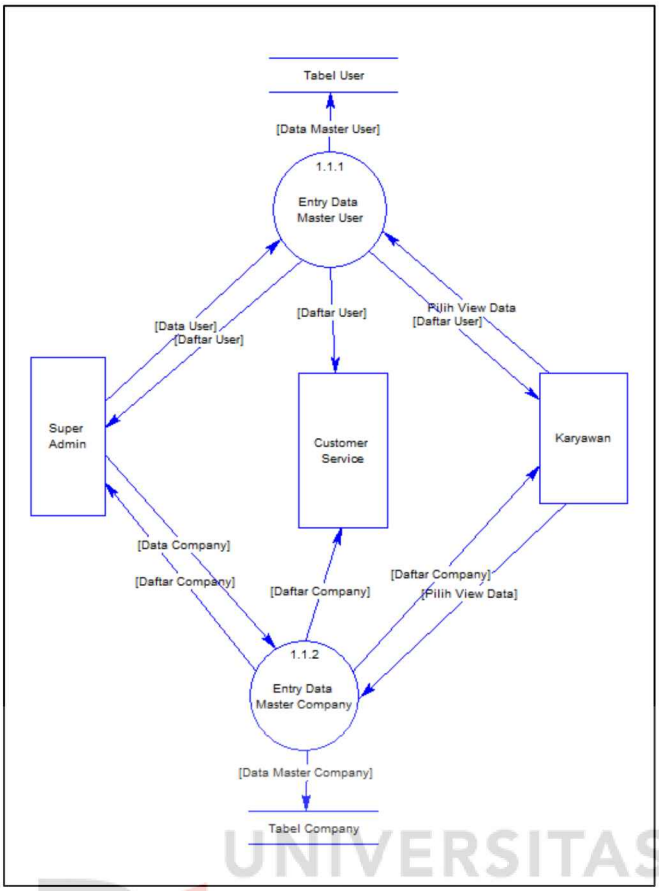
Gambar 4. 31 DFD Level O

2. Level 1

DFD level 1 merupakan penjabaran dari setiap proses yang ada pada DFD level 0. Proses-proses tersebut antara lain entry data *master*, entry data pendukung, dan entry data transaksi. Berikut merupakan gambar setiap proses DFD level 1.

2.1 Entry Data *Master*

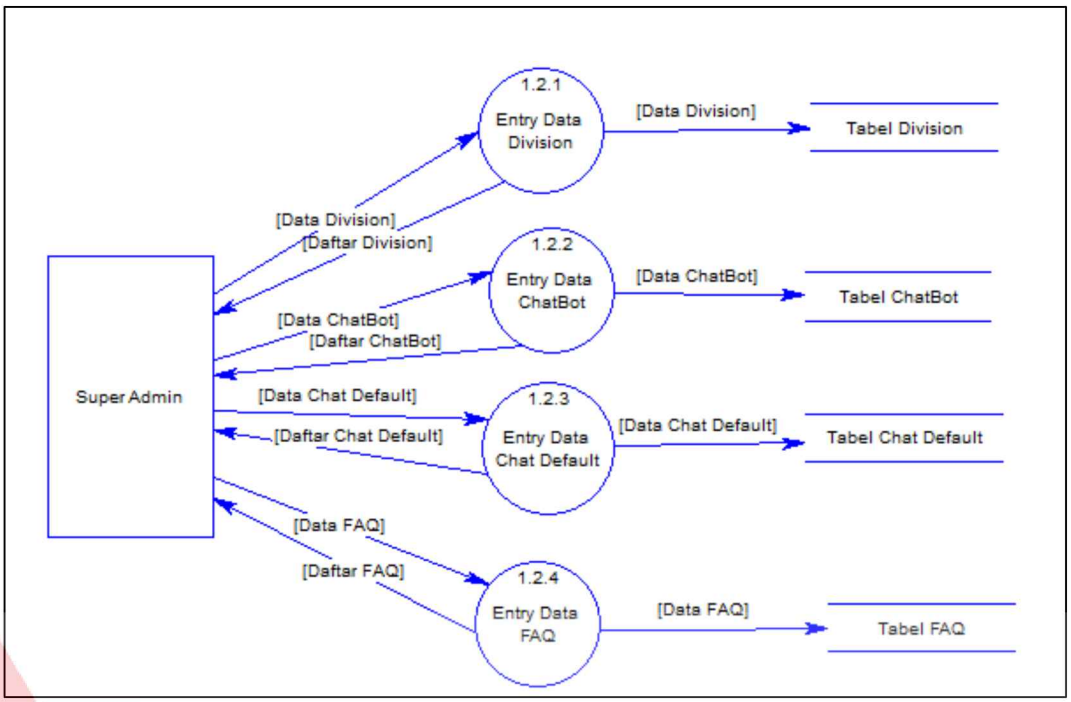
Gambar dibawah merupakan gambar dari DFD level 1 entry data *master*.



Gambar 4. 32 DFD Level 1 Entry Data *Master*

2.2 Entry Data Pendukung

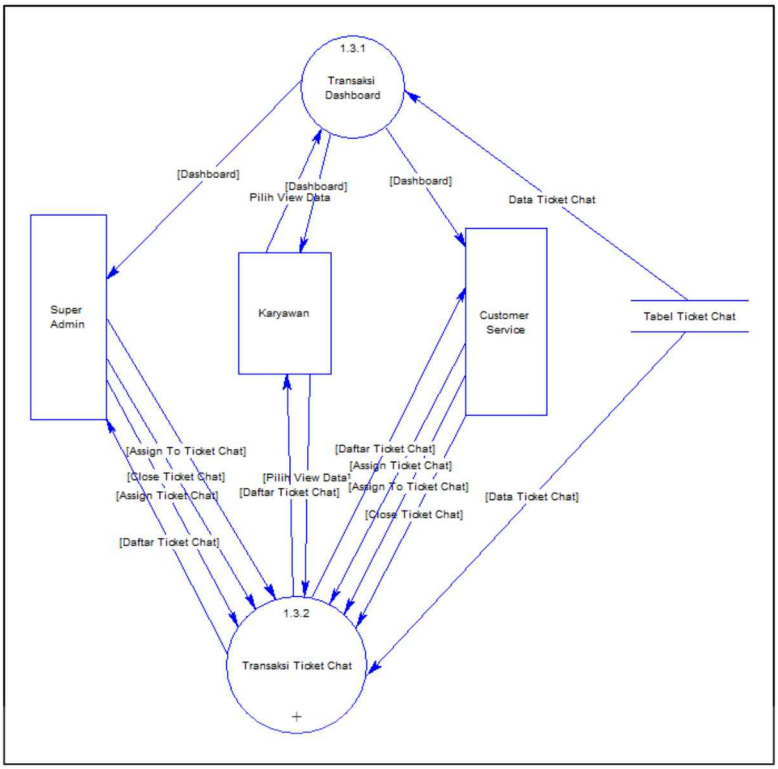
Gambar dibawah merupakan gambar dari DFD level 1 entry data pendukung.



Gambar 4. 33 DFD Level 1 Entry Data Pendukung

2.3 Entry Data Transaksi

Gambar dibawah merupakan gambar dari DFD level 1 entry data Transaksi.



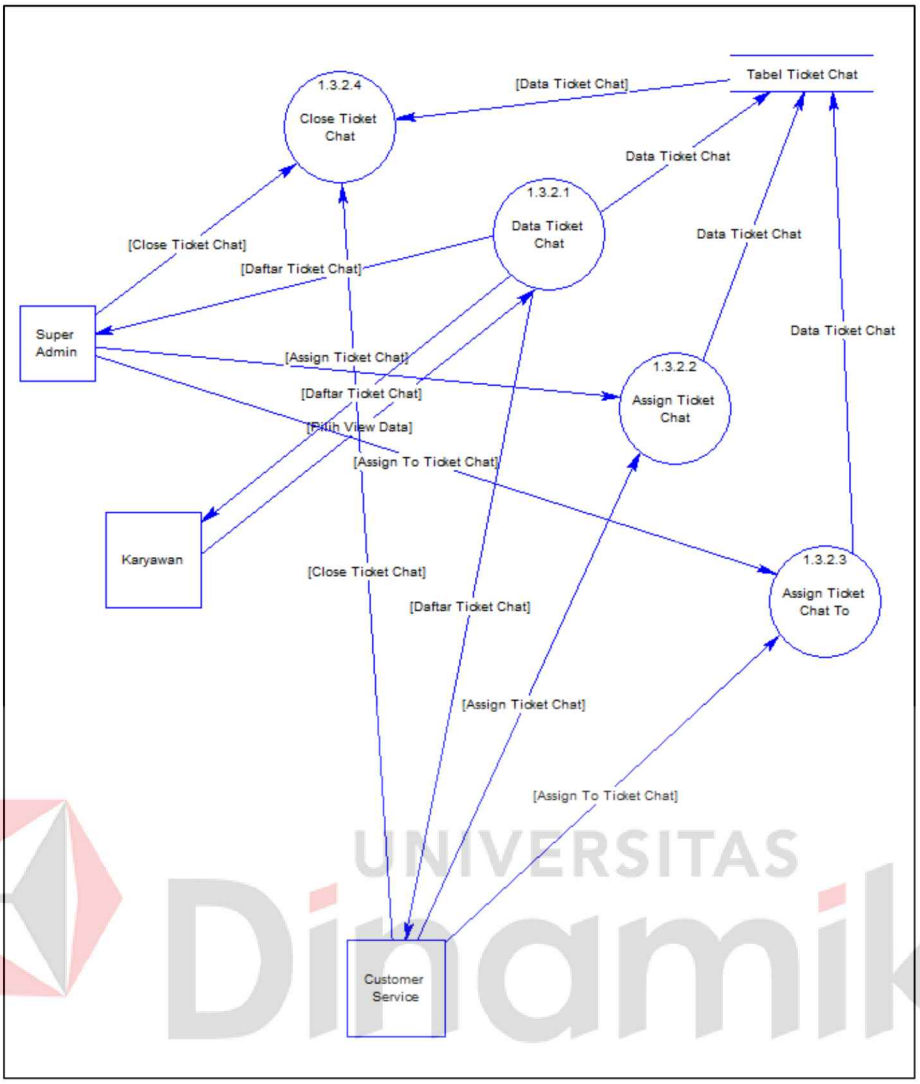
Gambar 4. 34 DFD Level 1 Entry Data Transaksi

3. Level 2

Pada DFD level 2 proses didapatkan dari penjabaran proses pada DFD level 1. DFD level 2 sendiri terdiri dari transaksi *Ticket Chat* yang merupakan penjabaran dari proses yang terdapat di dalam entry data transaksi.

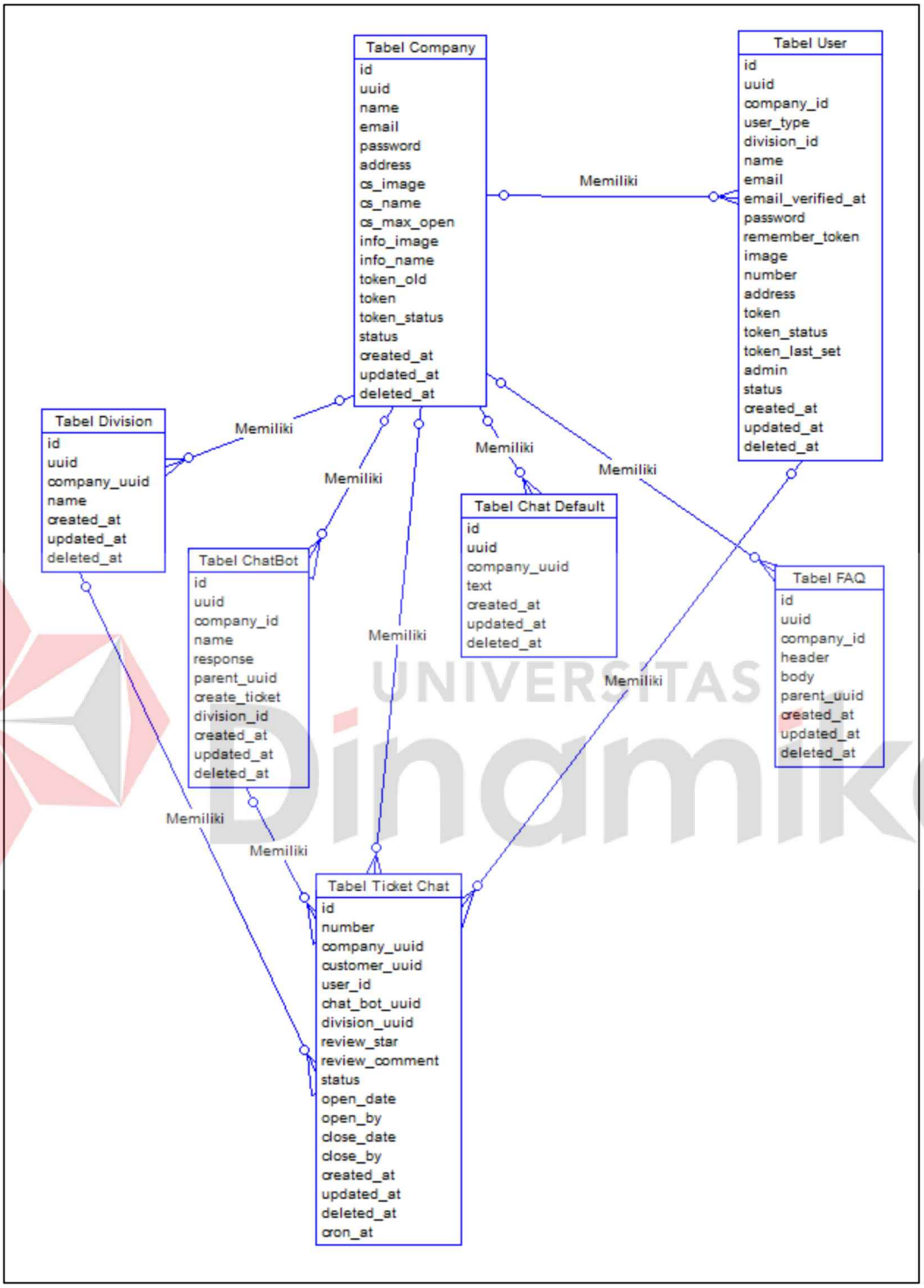
3.1 Transaksi *Ticket Chat*

Gambar dibawah merupakan gambar dari DFD level 2 transaksi *Ticket Chat*.



Gambar 4. 35 DFD Level 2 Transaksi *Ticket Chat*

4.10 CDM

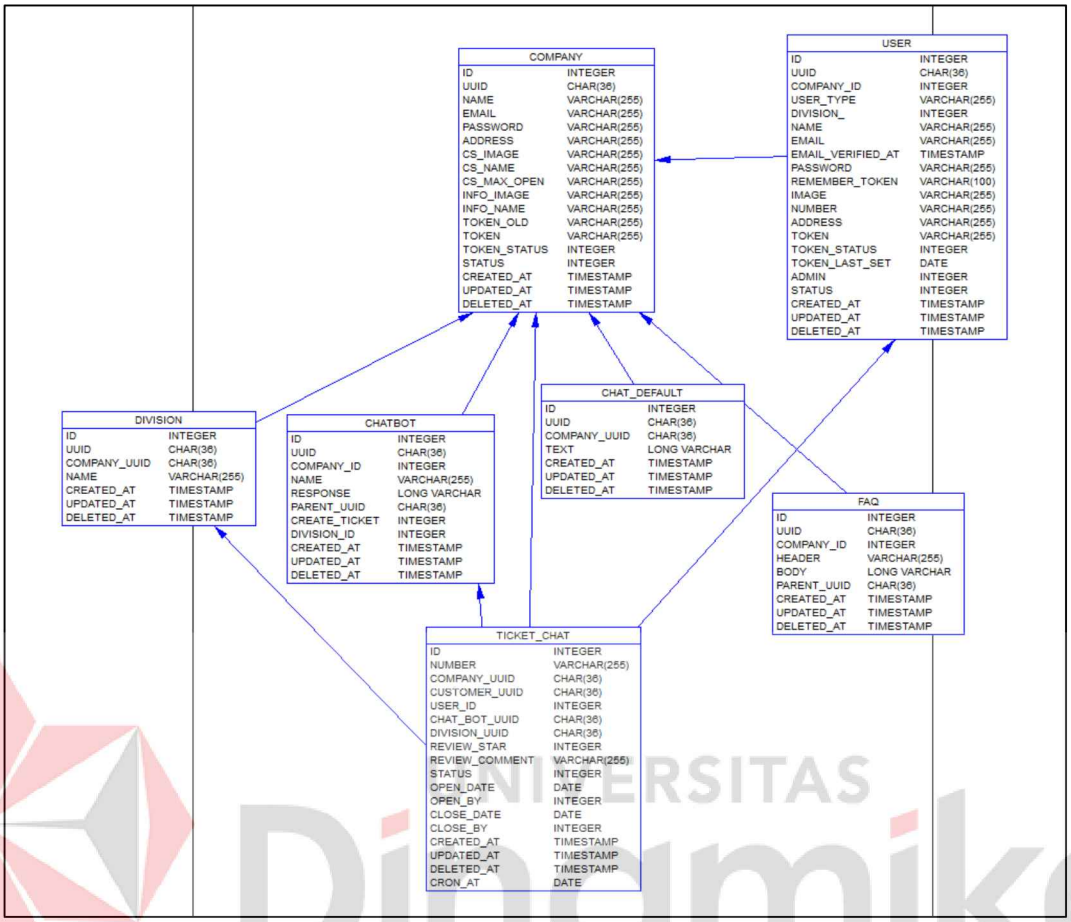


Gambar 4. 36 CDM

Gambar 4.36 menjelaskan tentang model CDM yang telah dibuat. Model CDM yang dibuat terdiri dari tujuh tabel. Tabel –tabel tersebut antara lain:

1. Tabel *Company*
2. Tabel *User*
3. Tabel *Division*
4. Tabel *ChatBot*
5. Tabel *Chat Default*
6. Tabel FAQ
7. Tabel *Ticket Chat*

4.11 PDM



Gambar 4. 37 PDM

Struktur tabel yang digunakan pada meliputi nama tabel, fungsi tabel, tipe data dan atribut seperti primary key dan foreign key. Berikut adalah struktur tabel:

A. Tabel *Master Company*

- Primary Key : id
- Foreign Key : -
- Fungsi tabel : Menyimpan data *master Company*

Table 4. 3 Struktur Tabel *Master Company*

No	Field	Tipe	Panjang	Keterangan
1	id	Integer		Id dari *Company*
2	uuid	Char	36	Uuid dari *Company*
3	email	Varchar	255	Email dari *Company*

No	Field	Tipe	Panjang	Keterangan
4	password	Varchar	255	Password milik *Company*
5	address	Varchar	255	Alamat dari *Company*
6	cs_image	Varchar	255	Gambar profile cs dari *Company*
7	cs_name	Varchar	255	Nama profile cs dari *Company*
8	cs_max_open	Varchar	255	Jumlah maksimal cs dapat menghandle ticket
9	info_image	Varchar	255	Gambar info dari *Company*
10	info_name	Varchar	255	Nama info dari *Company*
11	token_old	Varchar	255	Token yang didapat saat mendaftar
12	token	Varchar	255	Token yang didapat saat email telah diverifikasi
13	token_status	Integer		Status token setelah dan sebelum diverifikasi
14	status	Integer		Status *Publish* dari *Company*
15	created_at	Timestamp		Tanggal dan waktu data dibuat
16	updated_at	Timestamp		Tanggal dan wakktu data diubah
17	*deleted*_at	Timestamp		Tanggal dan waktu data dihapus

B. Tabel *Master User*

- Primary Key : id
- Foreign Key : *Company*_id, *Division*_id
- Fungsi tabel : Menyimpan data *master user*

Table 4. 4 Struktur Tabel *Master User*

No	Field	Tipe	Panjang	Keterangan
1	id	Integer		Id dari *user*
2	uuid	Char	36	Uuid dari *user*

No	Field	Tipe	Panjang	Keterangan
3	*Company*_id	Integer		Foreign key dari *Company* id yang terdaftar
4	*user*_type	Varchar	255	Tipe *user* yang dimiliki
5	*Division*_id	Integer		Foreign key dari *Division* id yang terdaftar
6	name	Varchar	255	Nama dari *user*
7	email	Varchar	255	Email dari *Company*
8	email_verified_at	Timestamp		Tanggal dan waktu email telah diverifikasi
9	password	Varchar	255	Password milik *Company*
10	remember_token	Varchar	100	Token yang diterima *user* saat mendaftar
11	image	Varchar	255	Foto profil dari *user*
12	number	Varchar	255	Nomor telepon dari *user*
13	address	Varchar	255	Alamat dari *Company*
14	token	Varchar	255	Token yang didapat saat email telah diverifikasi
15	token_status	Integer		Status token setelah dan sebelum diverifikasi
16	token_last_sent	Date		Tanggal token terakhir dikirim untuk penggantian password
17	admin	Integer		Menunjukan apakah *user* ini admin atau bukan
18	status	Integer		Status dari *user* ini
19	created_at	Timestamp		Tanggal dan waktu data dibuat
20	updated_at	Timestamp		Tanggal dan wakktu data diubah

21	*deleted*_at	Timestamp	Tanggal dan waktu data dihapus

C. Tabel *Division*

- Primary Key : id
- Foreign Key : *Company*_uuid
- Fungsi tabel : Menyimpan data *Division*

Table 4. 5 Struktur Tabel *Division*

No	Field	Tipe	Panjang	Keterangan
1	id	Integer		Id dari *Division*
2	uuid	Varchar	255	Uuid dari *Division*
3	*Company*_uuid	Char	36	Foreign key dari *Company* uuid yang terdaftar
4	name	Varchar	255	Nama dari *Division*
5	created_at	Timestamp		Tanggal dan waktu data dibuat
6	updated_at	Timestamp		Tanggal dan wakktu data diubah
7	*deleted*_at	Timestamp		Tanggal dan waktu data dihapus

D. Tabel *ChatBot*

- Primary Key : id
- Foreign Key : *Company*_id, parent_uuid, *Division*_id
- Fungsi tabel : Menyimpan data *ChatBot*

Table 4. 6 Struktur Tabel *ChatBot*

No	Field	Tipe	Panjang	Keterangan
1	id	Integer		Id dari *ChatBot*
2	uuid	Char	36	Uuid dari *ChatBot*
3	*Company*_id	Integer		Foreign key dari *Company* id yang terdaftar
4	name	Varchar	255	Nama dari *ChatBot*

No	Field	Tipe	Panjang	Keterangan
5	response	Text		Response dari *ChatBot*
6	parent_uuid	Varchar	255	Uuid dari parent *ChatBot*
7	create_ticket	Integer		Create ticket dibuat atau tidak
8	*Division*_id	Integer		Foreign key dari *Division* id yang terdaftar
9	created_at	Timestamp		Tanggal dan waktu data dibuat
10	updated_at	Timestamp		Tanggal dan wakktu data diubah
11	*delete*d_at	Timestamp		Tanggal dan waktu data dihapus

E. Tabel *Chat Default*

- Primary Key : id
- Foreign Key : *Company*_uuid
- Fungsi tabel : Menyimpan data *Chat Default*

Table 4. 7 Struktur Tabel *Chat Default*

No	Field	Tipe	Panjang	Keterangan
1	id	Integer		Id dari *Chat Default*
2	uuid	Char	36	Uuid dari *Chat Default*
3	*Company*_uuid	Char	36	Foreign key dari *Company* uuid yang terdaftar
4	text	Text		Isi text dari *Chat Default*
5	created_at	Timestamp		Tanggal dan waktu data dibuat
6	updated_at	Timestamp		Tanggal dan wakktu data diubah
7	*delete*d_at	Timestamp		Tanggal dan waktu data dihapus

F. Tabel FAQ

- Primary Key : id
- Foreign Key : *Company*_id
- Fungsi tabel : Menyimpan data faq

Table 4. 8 Struktur Tabel FAQ

No	Field	Tipe	Panjang	Keterangan
1	id	Integer		Id dari faq
2	uuid	Char	36	Uuid dari faq
3	*Company*_id	Integer		Foreign key dari *Company* id yang terdaftar
4	header	Varchar	255	Header atau judul dari FAQ
5	body	Text		Isi text dari FAQ
6	created_at	Timestamp		Tanggal dan waktu data dibuat
7	updated_at	Timestamp		Tanggal dan wakktu data diubah
8	*delete*d_at	Timestamp		Tanggal dan waktu data dihapus

G. Tabel *Ticket Chat*

- Primary Key : id
- Foreign Key : *Company*_uuid, *user*_id, *ChatBot*_uuid, *Division*_uuid, open_by, close_by
- Fungsi tabel : Menyimpan data *Ticket Chat*

Table 4. 9 Struktur Tabel *Ticket Chat*

No	Field	Tipe	Panjang	Keterangan
1	id	Integer		Id dari *Ticket Chat*
2	number	Varchar	255	Uuid dari *Ticket Chat*
3	*Company*_uuid	Char	36	Foreign key dari *Company* uuid yang terdaftar
4	customer_uuid	Char	36	Foreign key dari cutomer uuid yang terdaftar

No	Field	Tipe	Panjang	Keterangan
5	*user*_id	Integer		Foreign key dari *user* id yang terdaftar
6	chat_bot_uuid	Char	36	Foreign key dari *ChatBot* uuid yang terdaftar
7	*Division*_uuid	Char	36	Foreign key dari *Division* uuid yang terdaftar
8	review_star	Integer		Jumlah nilai review yang diberikan customer
9	review_comment	Varchar	255	Komentar yang diberikan customer
10	status	Integer		Status dari ticket
11	open_date	Date		Tanggal ticket diambil alih oleh *user* pada *Company*
12	open_by	Integer		Foreign key dari *user* id yang terdaftar yang mengambil alih ticket
13	close_date	Date		Tanggal ticket ditutup oleh *user* pada *Company*
14	close_by	Integer		Foreign key dari *user* id yang terdaftar yang menutup ticket
15	created_at	Timestamp		Tanggal dan waktu data dibuat
16	updated_at	Timestamp		Tanggal dan wakktu data diubah
17	*delete*d_at	Timestamp		Tanggal dan waktu data dihapus
18	cron_at	Date		Tanggal panglihan ticket ke *user* lain

4.12 Kode Program

Berikut merupakan kebutuhan sistem yang ada yang terdiri dari setiap bagian kode-kode pada laravel.

4.12.1 *Company*

Pada bagian *Company* terdapat beberapa fungsi-fungsi yang ditunjukan pada setiap gambar-gambar dibawah. Gambar 4.38 menjelaskan fungsi *show* data yang bertujuan untuk menampilkan data yang ada pada *database*. Gambar 4.39 menjelaskan fungsi *Insert* data yang bertujuan untuk menambahkan data baru ke *database*. Gambar 4.40 merupakan lanjutan fungsi dari gambar 3.39. Gambar 4.41 menjelaskan fungsi *Update* data yang bertujuan untuk mengubah data yang sudah ada pada *database*. Gambar 4.42 merupakan lanjutan fungsi dari gambar 4.41.

Gambar 4.43 menjelaskan fungsi *Publish Company* yang bertujuan untuk mengubah status dari *Company* menjadi *Publish*. Status ini sendiri berfungsi agar pengguna diluar dari perusahaan ini dapat melihat info dari perusahaan ini. Gambar 4.44 menjelaskan fungsi *UnPublish Company* yang bertujuan untuk mengubah status dari perusahaan menjadi *UnPublish*, yang nantinya membuat pengguna lain tidak dapat melihat info dari perusahaan ini.

```php
public function detail(Request $request)
  {
      $user = Auth::user();

      $company = Companies::where('id', $user->company_id)->first();

      if (!$company) {
        return response()->json([
          'code' => 404,
          'message' => 'Company not found.'
        ], 404);
      }

      return response()->json([
          'code' => 200,
          'company' => $company
      ], 200);
  }
```

Gambar 4. 38 *Code Company Show*

```php
public function create(Request $request) {
      $bucket = 'klikchat-dev';
      $endpoint = config('aws.endpoint');
      $path = str_replace('https://', 'https://'.$bucket.'.', $endpoint);

      $user = Auth::user();

      $company = Companies::where('id', $user->company_id)->firstOrFail();

      if (!$company) {
        return response()->json([
          'code' => 404,
          'message' => 'Company not found.'
        ], 404);
      }

      $curlType = 1;

      $date = Carbon::now();

      $company->name = $request->name;
      $company->address = $request->address;
      $company->cs_name = $request->cs_name;
      $company->cs_max_open = $request->cs_max_open;
      $company->info_name = $request->info_name;

      // CS IMAGE
      $cs_image = '';
      if ($request->hasFile('cs_image')) {
        $validator = Validator::make($request->all(), [
          'cs_image' => 'mimes:jpeg,jpg,png'
        ]);

        if ($validator->fails()) {
            return response()->json([
                'code' => 400,
                'message' => $validator->errors()
            ], 400);
        }

        $file = $request->file('cs_image');
        $name = Str::random(10).''.$date->timestamp.'.'.$request->file('cs_image')->getClientOriginalExtension();
        $file_path = $file->getPathName();

        $s3 = AWS::createClient('s3');
        if (config('app.env') == 'production') {
          $upload = $s3->putObject(array(
              'ACL'                 => 'public-read',
              'ServerSideEncryption'  => 'AES256',
              'Bucket'              => $bucket,
              'Key'                 => 'business/'.$company->uuid.'/'.$name,
              'SourceFile'          => $file_path,
              'ContentType'         => $file->getMimeType(),
          ));
        } else {
          $upload = $s3->putObject(array(
              'ACL'                 => 'public-read',
              'StorageClass'        => 'STANDARD',
              'Bucket'              => $bucket,
              'Key'                 => 'business/'.$company->uuid.'/'.$name,
              'SourceFile'          => $file_path,
              'ContentType'         => $file->getMimeType(),
          ));
        }
        $cs_image = $path.'/'.'business/'.$company->uuid.'/'.$name;

      } else {
        $cs_image = $request->cs_imageDefault;
      }

      $company->cs_image = $cs_image;

      // INFO IMAGE
      $info_image = '';
      if ($request->hasFile('info_image')) {
        $validator = Validator::make($request->all(), [
          'info_image' => 'mimes:jpeg,jpg,png'
        ]);

        if ($validator->fails()) {
            return response()->json([
                'code' => 400,
                'message' => $validator->errors()
            ], 400);
        }
```

Gambar 4. 39 *Code Company insert*

```
        $file = $request->file('info_image');
        $name = Str::random(10).''.$date->timestamp.'.'.$request->file('info_image')-
>getClientOriginalExtension();
        $file_path = $file->getPathName();

        $s3 = AWS::createClient('s3');
        if (config('app.env') == 'production') {
          $upload = $s3->putObject(array(
              'ACL'                  => 'public-read',
              'ServerSideEncryption' => 'AES256',
              'Bucket'               => $bucket,
              'Key'                  => 'business/'.$company->uuid.'/'.$name,
              'SourceFile'           => $file_path,
              'ContentType'          => $file->getMimeType(),
          ));
        } else {
          $upload = $s3->putObject(array(
              'ACL'                  => 'public-read',
              'StorageClass'         => 'STANDARD',
              'Bucket'               => $bucket,
              'Key'                  => 'business/'.$company->uuid.'/'.$name,
              'SourceFile'           => $file_path,
              'ContentType'          => $file->getMimeType(),
          ));
        }
        $info_image = $path.'/'.'business/'.$company->uuid.'/'.$name;

      } else {
        $info_image = $request->info_imageDefault;
      }

      $company->info_image = $info_image;

      // CURL START HERE
      if ($curlType == 1) {
          $response = Curl::to(config('api.url').'/company')
                          ->withContentType('application/json')
                          ->withData(array(
                            'uuid' => $company->uuid,
                            'token' => $company->token,
                            'csName' => $company->cs_name,
                            'csImage' => $company->cs_image,
                            'infoName' => $company->info_name,
                            'infoImage' => $company->info_image,
                          ))
                          ->asJson(true)
                          ->post();
      }

      if ($response) {
          if (isset($response->error)) {
              return response()->json([
                'code' => $response->error->code,
                'message' => $response->error->message,
            ], $response->error->code);
          }
      }
      // CURL END HERE

      $company->save();

      $user->name = $company->name;
      $user->image = $company->cs_image;
      $user->save();

      return response()->json([
        'company' => $company,
        'code' => 200,
        'data' => 'Data saved.',
      ], 200);
  }
```

Gambar 4. 40 *Code Company Insert (*lanjutan*)*

```php
public function update(Request $request, $company_uuid)
  {
      $bucket = 'klikchat-dev';
      $endpoint = config('aws.endpoint');
      $path = str_replace('https://', 'https://'.$bucket.'.', $endpoint);

      $user = Auth::user();

      $company = Companies::where('id', $user->company_id)->firstOrFail();

      if (!$company) {
        return response()->json([
          'code' => 404,
          'message' => 'Company not found.'
        ], 404);
      }

      $curlType = 1;

      $date = Carbon::now();

      $company->name = $request->name;
      $company->address = $request->address;
      $company->cs_name = $request->cs_name;
      $company->cs_max_open = $request->cs_max_open;
      $company->info_name = $request->info_name;

      // CS IMAGE
      $cs_image = '';
      if ($request->hasFile('cs_image')) {
        $validator = Validator::make($request->all(), [
          'cs_image' => 'mimes:jpeg,jpg,png'
        ]);

        if ($validator->fails()) {
            return response()->json([
                'code' => 400,
                'message' => $validator->errors()
            ], 400);
        }

        $file = $request->file('cs_image');
        $name = Str::random(10).''.$date->timestamp.'.'.$request->file('cs_image')->getClientOriginalExtension();
        $file_path = $file->getPathName();

        $s3 = AWS::createClient('s3');
        if (config('app.env') == 'production') {
          $upload = $s3->putObject(array(
              'ACL'                 => 'public-read',
              'ServerSideEncryption'  => 'AES256',
              'Bucket'              => $bucket,
              'Key'                 => 'business/'.$company->uuid.'/'.$name,
              'SourceFile'          => $file_path,
              'ContentType'         => $file->getMimeType(),
          ));
        } else {
          $upload = $s3->putObject(array(
              'ACL'                 => 'public-read',
              'StorageClass'        => 'STANDARD',
              'Bucket'              => $bucket,
              'Key'                 => 'business/'.$company->uuid.'/'.$name,
              'SourceFile'          => $file_path,
              'ContentType'         => $file->getMimeType(),
          ));
        }
        $cs_image = $path.'/'.'business/'.$company->uuid.'/'.$name;

      } else {
        $cs_image = $request->cs_imageDefault;
      }

      $company->cs_image = $cs_image;

      // INFO IMAGE
      $info_image = '';
      if ($request->hasFile('info_image')) {
        $validator = Validator::make($request->all(), [
          'info_image' => 'mimes:jpeg,jpg,png'
        ]);

        if ($validator->fails()) {
            return response()->json([
                'code' => 400,
                'message' => $validator->errors()
            ], 400);
        }
```

Gambar 4. 41 *Code Company Update*

```
        $file = $request->file('info_image');
        $name = Str::random(10).''.$date->timestamp.'.'.$request->file('info_image')-
>getClientOriginalExtension();
        $file_path = $file->getPathName();

        $s3 = AWS::createClient('s3');
        if (config('app.env') == 'production') {
          $upload = $s3->putObject(array(
              'ACL'                 => 'public-read',
              'ServerSideEncryption'  => 'AES256',
              'Bucket'              => $bucket,
              'Key'                 => 'business/'.$company->uuid.'/'.$name,
              'SourceFile'          => $file_path,
              'ContentType'         => $file->getMimeType(),
          ));
        } else {
          $upload = $s3->putObject(array(
              'ACL'                 => 'public-read',
              'StorageClass'        => 'STANDARD',
              'Bucket'              => $bucket,
              'Key'                 => 'business/'.$company->uuid.'/'.$name,
              'SourceFile'          => $file_path,
              'ContentType'         => $file->getMimeType(),
          ));
        }
        $info_image = $path.'/'.'business/'.$company->uuid.'/'.$name;

      } else {
        $info_image = $request->info_imageDefault;
      }

      $company->info_image = $info_image;

      // CURL START HERE
      if ($curlType == 1) {
          $response = Curl::to(config('api.url').'/company/'.$company->uuid)
                          ->withContentType('application/json')
                          ->withHeaders(array(
                                'Authorization: '.$company->token.''
                          ))
                          ->withData(array(
                            'uuid' => $company->uuid,
                            'token' => $company->token,
                            'csName' => $company->cs_name,
                            'csImage' => $company->cs_image,
                            'infoName' => $company->info_name,
                            'infoImage' => $company->info_image,
                          ))
                          ->asJson(true)
                          ->put();
      }

      if ($response) {
          if (isset($response->error)) {
              return response()->json([
                'code' => $response->error->code,
                'message' => $response->error->message,
            ], $response->error->code);
          }
      }
      // CURL END HERE

      $company->save();

      $user->name = $company->name;
      $user->image = $company->cs_image;
      $user->save();

      return response()->json([
        'company' => $company,
        'code' => 200,
        'data' => 'Data saved.',
      ], 200);
  }
```


Gambar 4. 42 *Code Company Update (*lanjutan)

```
public function publish(Request $request)
  {
      $user = Auth::user();

      $company = Companies::where('id', $user->company_id)->firstOrFail();

      if (!$company) {
        return response()->json([
          'code' => 404,
          'message' => 'Company not found.'
        ], 404);
      }

      $response = Curl::to(config('api.url').'/company/'.$company->uuid)
                      ->withContentType('application/json')
                      ->withHeaders(array(
                            'Authorization: '.$company->token.'',
                            'suspend: False'
                      ))
                      ->asJson(true)
                      ->delete();

      if ($response) {
          if (isset($response->error)) {
              return response()->json([
                  'code' => $response->error->code,
                  'message' => $response->error->message,
              ], $response->error->code);
          }
      }

      $company->status = 1;
      $company->save();

      $success['message'] = 'Company published';
      $success['code'] =  200;

      return response()->json([
          'success' => $success
      ], 200);
  }
```

Gambar 4. 43 *Code Company Publish*

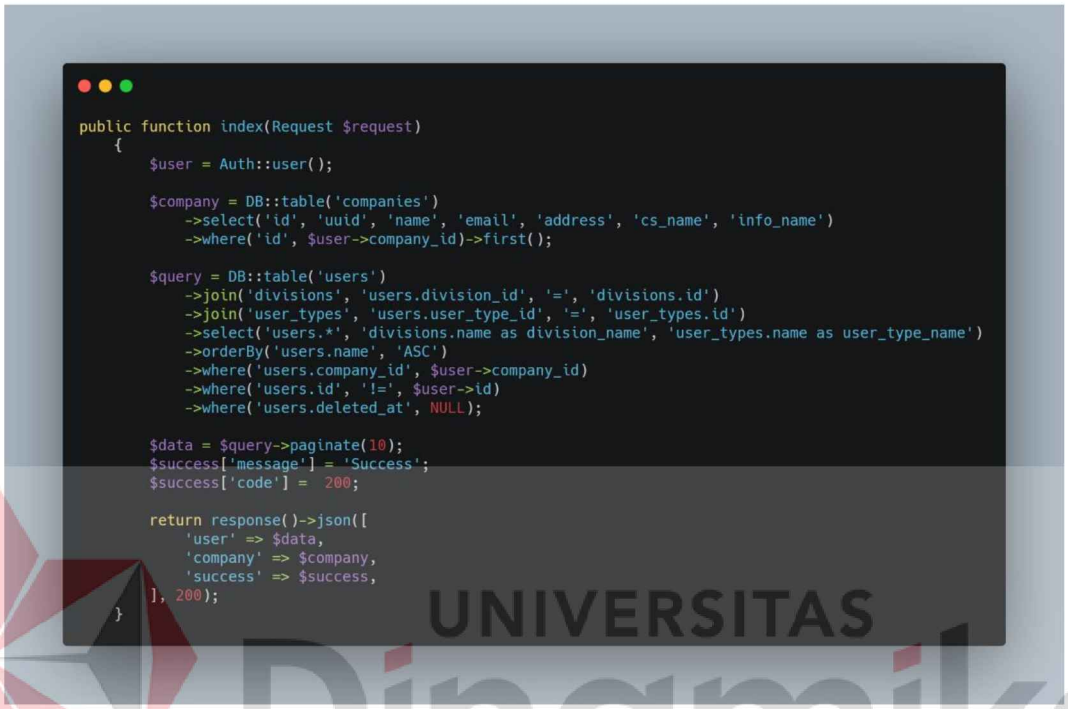
```
public function unpublish(Request $request)
  {
      $user = Auth::user();

      $company = Companies::where('id', $user->company_id)->firstOrFail();

      if (!$company) {
        return response()->json([
          'code' => 404,
          'message' => 'Company not found.'
        ], 404);
      }

      $response = Curl::to(config('api.url').'/company/'.$company->uuid)
                    ->withContentType('application/json')
                    ->withHeaders(array(
                          'Authorization: '.$company->token.'',
                          'suspend: True'
                    ))
                    ->asJson(true)
                    ->delete();

      if ($response) {
          if (isset($response->error)) {
              return response()->json([
                  'code' => $response->error->code,
                  'message' => $response->error->message,
              ], $response->error->code);
          }
      }

      $company->status = 0;
      $company->save();

      $success['message'] = 'Company unpublished';
      $success['code'] =  200;

      return response()->json([
          'success' => $success
      ], 200);
  }
```

Gambar 4. 44 *Code Company UnPublish*

4.12.2 *User*

Pada bagian *user* terdapat empat fungsi yang ditunjukan pada setiap gambar dibawah. Gambar 4.45 menjelaskan fungsi *show* data yang bertujuan untuk menampilkan data yang ada pada *database*. Gambar 4.46 menjelaskan fungsi *Insert* data yang bertujuan untuk menambahkan data baru ke *database*. Gambar 4.47

menjelaskan fungsi *Update* data yang bertujuan untuk mengubah data yang sudah ada pada *database*. Gambar 4.48 menjelaskan fungsi *delete* data yang bertujuan untuk menghapus data tertentu yang ada pada *database*.



```php
public function index(Request $request)
    {
        $user = Auth::user();

        $company = DB::table('companies')
            ->select('id', 'uuid', 'name', 'email', 'address', 'cs_name', 'info_name')
            ->where('id', $user->company_id)->first();

        $query = DB::table('users')
            ->join('divisions', 'users.division_id', '=', 'divisions.id')
            ->join('user_types', 'users.user_type_id', '=', 'user_types.id')
            ->select('users.*', 'divisions.name as division_name', 'user_types.name as user_type_name')
            ->orderBy('users.name', 'ASC')
            ->where('users.company_id', $user->company_id)
            ->where('users.id', '!=', $user->id)
            ->where('users.deleted_at', NULL);

        $data = $query->paginate(10);
        $success['message'] = 'Success';
        $success['code'] =  200;

        return response()->json([
            'user' => $data,
            'company' => $company,
            'success' => $success,
        ], 200);
    }
```

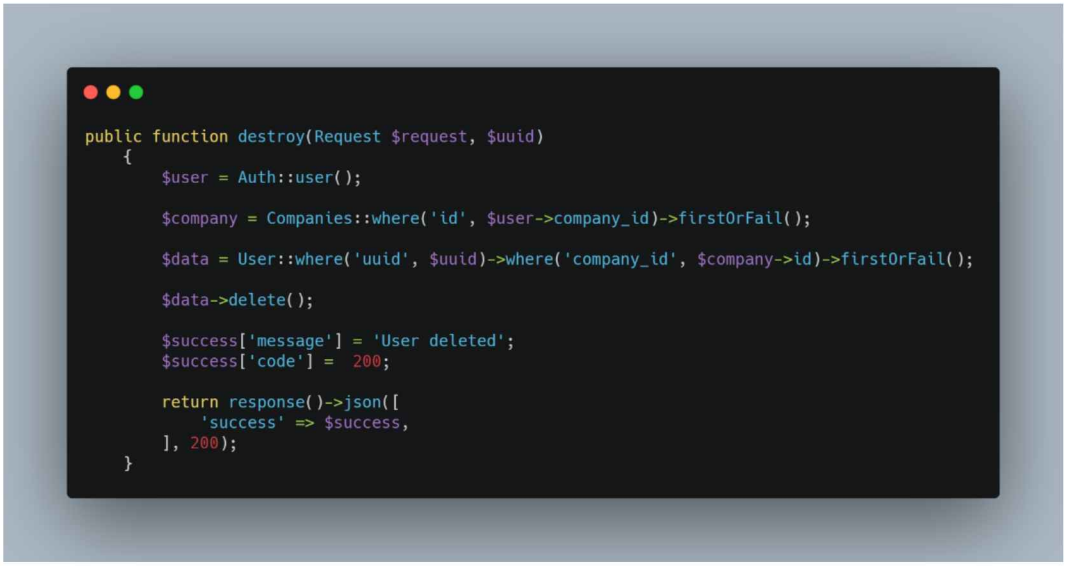
Gambar 4. 45 *Code User Show*

```php
public function store(Request $request)
    {
        $user = Auth::user();

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        $this->validate($request, [
            'name' => 'required|string|max:255',
            'number' => 'required|string|max:15|unique:users',
            'email' => 'required|string|max:50|email|unique:users',
            'address' => 'required|string|max:255',
            'division_id' => 'required|integer',
            'user_type_id' => 'required|integer',
        ]);

        $division = Division::where('id', $request->division_id)->firstOrFail();

        if ($division->company_uuid != $company->uuid) {
            $success['message'] = 'Division was not found in this company';
            $success['code'] =  404;

            return response()->json([
                'success' => $success,
            ], 404);
        }

        $user_type = UserType::where('id', $request->user_type_id)->firstOrFail();

        $data = User::create([
            'uuid' => Str::random(17),
            'company_id' => $user->company_id,
            'user_type_id' => $request->input('user_type_id'),
            'division_id' => $request->input('division_id'),
            'name' => $request->input('name'),
            'email' => $request->input('email'),
            'password' => '',
            'number' => $request->input('number'),
            'address' => $request->input('address'),
        ]);

        $success['message'] = 'User created';
        $success['code'] =  200;

        return response()->json([
            'user' => $data,
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 46 *Code User Insert*

```php
public function update(Request $request, $uuid)
    {
        $user = Auth::user();

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        $data = User::where('uuid', $uuid)->where('company_id', $company->id)->firstOrFail();

        if ($data->company_id != $company->id) {
            $success['message'] = 'User was not found in this company';
            $success['code'] =  404;

            return response()->json([
                'success' => $success,
            ], 404);
        }

        $this->validate($request, [
            'name' => 'required|string|max:255',
            'number' => 'required|string|max:15|unique:users,number,'.$data->id.'',
            'email' => 'required|string|max:50|email|unique:users,email,'.$data->id.'',
            'address' => 'required|string|max:255',
            'division_id' => 'required|integer|max:20',
            'user_type_id' => 'required|integer|max:20',
        ]);

        $division = Division::where('id', $request->division_id)->firstOrFail();

        if ($division->company_uuid != $company->uuid) {
            $success['message'] = 'Division was not found in this company';
            $success['code'] =  404;

            return response()->json([
                'success' => $success,
            ], 404);
        }

        $user_type = UserType::where('id', $request->user_type_id)->firstOrFail();

        $data->update($request->all());

        $success['message'] = 'User updated';
        $success['code'] =  200;

        return response()->json([
            'user' => $data,
            'success' => $success,
        ], 200);
    }
```


Gambar 4. 47 *Code User Update*



```
public function destroy(Request $request, $uuid)
    {
        $user = Auth::user();

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        $data = User::where('uuid', $uuid)->where('company_id', $company->id)->firstOrFail();

        $data->delete();

        $success['message'] = 'User deleted';
        $success['code'] =  200;

        return response()->json([
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 48 *Code User Delete*

4.12.3 *Division*

Pada bagian *Division* juga terdapat empat fungsi yang ditunjukan pada setiap gambar dibawah. Gambar 4.49 menjelaskan fungsi *show* data *Division* yang bertujuan untuk menampilkan data yang ada pada *database*. Gambar 4.50 menjelaskan fungsi *Insert* data *Division* yang bertujuan untuk menambahkan data baru ke *database*. Gambar 4.51 menjelaskan fungsi *Update* data *Division* yang bertujuan untuk mengubah data yang sudah ada pada *database*. Gambar 4.52 menjelaskan fungsi *delete* data *Division* yang bertujuan untuk menghapus data tertentu yang ada pada *database*.

```php
public function index($currentPage)
  {
      $user = Auth::user();

      $company = Companies::where('id', $user->company_id)->firstOrFail();

      $query = Division::orderBy('name', 'ASC');
      $query->where('company_uuid', $company->uuid);

      Paginator::currentPageResolver(function() use ($currentPage) {
          return $currentPage;
      });

      $data = $query->paginate(10);
      $success['message'] = 'Success';
      $success['code'] =  200;

      return response()->json([
          'division' => $data,
          'success' => $success,
      ], 200);
  }
```

Gambar 4. 49 *Code Division Show*


```php
public function store(Request $request)
  {
      $user = Auth::user();

      $company = Companies::where('id', $user->company_id)->firstOrFail();

      $this->validate($request, [
          'name' => 'required|string|max:255',
      ]);

      $data = Division::create([
          'uuid' => Str::random(17),
          'company_uuid' => $company->uuid,
          'name' => $request->name,
      ]);

      $success['message'] = 'Division created';
      $success['code'] =  200;

      return response()->json([
        'division' => $data,
        'success' => $success,
      ], 200);
  }
```

Gambar 4. 50 *Code Division Insert*

```php
public function update(Request $request, $uuid)
  {
      $user = Auth::user();

      $data = Division::where('uuid', $uuid)->firstOrFail();

      $this->validate($request, [
          'name' => 'required|string|max:255',
      ]);

      $company = Companies::where('id', $user->company_id)->firstOrFail();

      if ($data->company_uuid != $company->uuid) {
          return response()->json([
              'code' => 404,
              'message' => 'Division was not found in this company'
          ], 404);
      }

      $data->update($request->all());

      $success['message'] = 'Division updated';
      $success['code'] =  200;

      return response()->json([
          'division' => $data,
          'success' => $success,
      ], 200);
  }
```

Gambar 4. 51 *Code Division Update*

```php
public function destroy(Request $request, $uuid)
  {
      $user = Auth::user();

      $company = Companies::where('id', $user->company_id)->firstOrFail();

      $division = Division::where('uuid', $uuid)->firstOrFail();

      if ($division->company_uuid != $company->uuid) {
          return response()->json([
              'code' => 404,
              'message' => 'Division was not found in this company'
          ], 404);
      }

      $division->delete();

      $success['message'] = 'Division deleted';
      $success['code'] =  200;

      return response()->json([
          'success' => $success,
      ], 200);
  }
```

Gambar 4. 52 *Code Division Delete*

4.12.4 *ChatBot*

Pada bagian *ChatBot* terdapat empat fungsi yang ditunjukan pada setiap gambar dibawah. Gambar 4.53 menjelaskan fungsi *show* data *ChatBot* yang bertujuan untuk menampilkan data yang ada pada *database*. Gambar 4.54 menjelaskan fungsi *Insert* data *ChatBot* yang bertujuan untuk menambahkan data baru ke *database*. Gambar 4.55 menjelaskan fungsi *Update* data *ChatBot* yang bertujuan untuk mengubah data yang sudah ada pada *database*. Gambar 4.56 menjelaskan fungsi *delete* data *ChatBot* yang bertujuan untuk menghapus data tertentu yang ada pada *database*.

```
public function index($page)
    {
        $user = Auth::user();

        $query = ChatBot::with(['division'])
            ->where('company_id', $user->company_id)
            ->where('parent_uuid', NULL);

        $data = $query->get();

        return fractal()
            ->collection($data, new ChatBotTransformer())
            ->serializeWith(new KcArraySerializer())
            ->toArray();
    }
```

Gambar 4. 53 *Code ChatBot Show*

```php
public function store(Request $request)
    {
        $user = Auth::user();

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        $this->validate($request, [
            'name' => 'required|string|max:255',
            'response' => 'required|string|max:255',
            'division_id' => 'required|integer|max:20',
        ]);

        $division = Division::where('id', $request->division_id)
            ->firstOrFail();

        if ($division->company_uuid != $company->uuid) {
            return response()->json([
                'code' => 404,
                'message' => 'Division was not found in this company'
            ], 404);
        }

        $data = ChatBot::create([
            'uuid' => Str::random(17),
            'company_id' => $user->company_id,
            'name' => $request->input('name'),
            'response' => $request->input('response'),
            'parent_uuid' => $request->input('parent_uuid'),
            'division_id' => $request->input('division_id'),
            'create_ticket' => 1
        ]);

        $success['message'] = 'Chatbot created';
        $success['code'] =  200;

        return response()->json([
            'chatbot' => $data,
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 54 *Code ChatBot Insert*

```
public function update(Request $request, $uuid)
    {
        $user = Auth::user();

        $data = ChatBot::where('uuid', $uuid)
            ->where('company_id', $user->company_id)
            ->firstOrFail();

        $this->validate($request, [
            'name' => 'required|string|max:255',
            'response' => 'required|string|max:255',
        ]);

        $data->update($request->all());

        $success['message'] = 'Chatbot updated';
        $success['code'] =  200;

        return response()->json([
            'chatbot' => $data,
            'success' => $success,
        ], 200);
    }
```

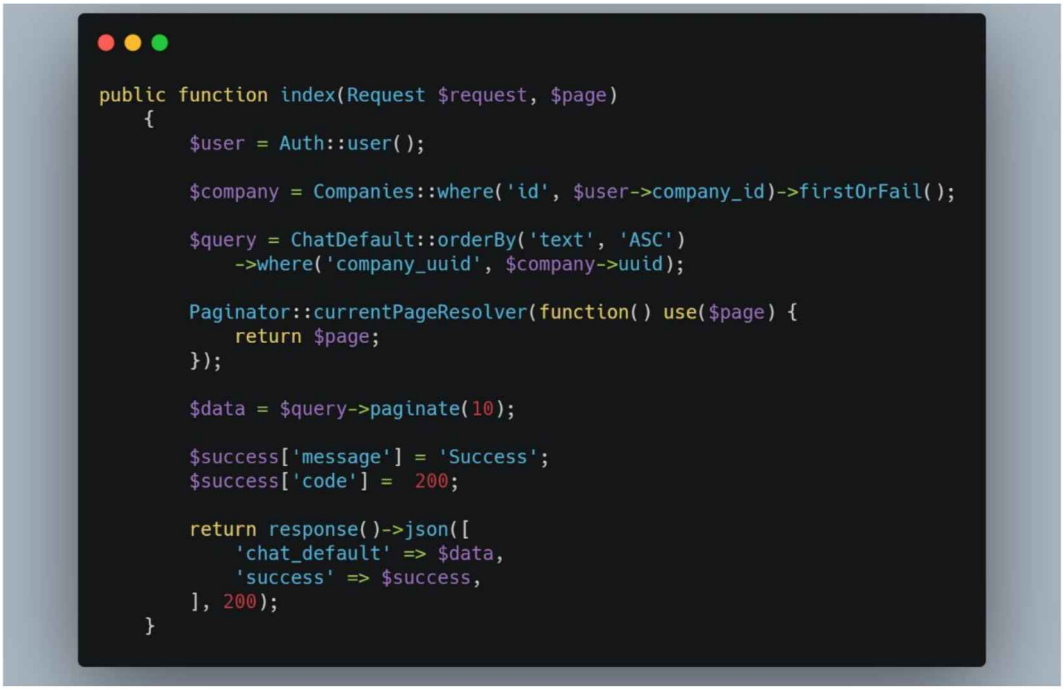

Gambar 4. 55 *Code ChatBot Update*

```
public function destroy(Request $request, $uuid)
    {
        $user = Auth::user();

        $chatBot = ChatBot::where('uuid', $uuid)
            ->where('company_id', $user->company_id)
            ->firstOrFail();

        if ($chatBot->company_id != $user->company_id) {
            return response()->json([
                'code' => 404,
                'message' => 'Not found.'
            ], 404);
        }

        $chatBot->delete();

        $success['message'] = 'Chatbot deleted';
        $success['code'] =  200;

        return response()->json([
            'success' => $success,
        ], 200);
    }
```


Gambar 4. 56 *Code ChatBot Delete*

4.12.5 *Chat Default*

Pada bagian *Chat Default* juga terdapat empat fungsi yang ditunjukan pada setiap gambar dibawah. Gambar 4.57 menjelaskan fungsi *show* data yang bertujuan untuk menampilkan data yang ada pada *database*. Gambar 4.58 menjelaskan fungsi *Insert* data yang bertujuan untuk menambahkan data baru ke *database*. Gambar 4.59 menjelaskan fungsi *Update* data yang bertujuan untuk mengubah data yang sudah ada pada *database*. Gambar 4.60 menjelaskan fungsi *delete* data yang bertujuan untuk menghapus data tertentu yang ada pada *database*.



```php
public function index(Request $request, $page)
    {
        $user = Auth::user();

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        $query = ChatDefault::orderBy('text', 'ASC')
            ->where('company_uuid', $company->uuid);

        Paginator::currentPageResolver(function() use($page) {
            return $page;
        });

        $data = $query->paginate(10);

        $success['message'] = 'Success';
        $success['code'] =  200;

        return response()->json([
            'chat_default' => $data,
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 57 *Code Chat Default Show*

```php
public function store(Request $request)
    {
        $user = Auth::user();

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        $this->validate($request, [
            'text' => 'required|string|max:255',
        ]);

        $data = ChatDefault::create([
            'uuid' => Str::random(17),
            'company_uuid' => $company->uuid,
            'text' => $request->input('text'),
        ]);

        $success['message'] = 'Chat default created';
        $success['code'] =  200;

        return response()->json([
            'chat_default' => $data,
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 58 *Code Chat Default Insert*

```php
public function update(Request $request, $uuid)
    {
        $user = Auth::user();

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        $data = ChatDefault::where('uuid', $uuid)->firstOrFail();

        $this->validate($request, [
            'text' => 'required|string|max:255',
        ]);

        if ($data->company_uuid != $company->uuid) {
            return response()->json([
                'code' => 404,
                'message' => 'Chat default was not found in this company'
            ], 404);
        }

        $data->update($request->all());

        $success['message'] = 'Chat default updated';
        $success['code'] =  200;

        return response()->json([
            'chat_default' => $data,
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 59 *Code Chat Default Update*

```php
public function destroy(Request $request, $uuid)
    {
        $user = Auth::user();

        $company = Companies::where('id', $user->company_id)->first();

        $chatDefault = ChatDefault::where('uuid', $uuid)->firstOrFail();

        if ($chatDefault->company_uuid != $company->uuid) {
            return response()->json([
                'code' => 404,
                'message' => 'Chat default was not found in this company'
            ], 404);
        }

        $chatDefault->delete();

        $success['message'] = 'Chat default deleted';
        $success['code'] =  200;

        return response()->json([
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 60 *Code Chat Default Delete*

4.12.6 FAQ

Pada bagian faq terdapat juga empat fungsi yang ditunjukan pada setiap gambar-gambar dibawah. Gambar 4.61 menjelaskan fungsi *show* data faq yang bertujuan untuk menampilkan data yang ada pada *database*. Gambar 4.62 menjelaskan fungsi *Insert* data faq yang bertujuan untuk menambahkan data baru ke *database*. Gambar 4.63 menjelaskan fungsi *Update* data faq yang bertujuan untuk mengubah data yang sudah ada pada *database*. Gambar 4.64 menjelaskan fungsi *delete* data faq yang bertujuan untuk menghapus data tertentu yang ada pada *database*.

```php
public function index()
    {
        $user = Auth::user();

        $data = Faq::where('company_id', $user->company_id)
            ->where('parent_uuid', NULL)
            ->where('company_id', $user->company_id)
            ->get();

        return fractal()
            ->collection($data, new FaqTransformer())
            ->serializeWith(new KcArraySerializer())
            ->toArray();
    }
```

Gambar 4. 61 *Code* FAQ *Show*

```php
public function store(Request $request)
    {
        $user = Auth::user();

        $this->validate($request, [
            'header' => 'required|string|max:255',
            'body' => 'required|string|max:255',
        ]);

        if ($request->parent_uuid != NULL) {
            $parent_faq = Faq::where('uuid', $request->parent_uuid)->firstOrFail();
        }

        $data = Faq::create([
            'uuid' => Str::random(17),
            'company_id' => $user->company_id,
            'header' => $request->input('header'),
            'body' => $request->input('body'),
            'parent_uuid' => $request->input('parent_uuid'),
        ]);

        $success['message'] = 'Faq created';
        $success['code'] =  200;

        return response()->json([
            'faq' => $data,
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 62 *Code* FAQ *Insert*

```php
public function update(Request $request, $uuid)
    {
        $user = Auth::user();

        $data = Faq::where('uuid', $uuid)->firstOrFail();

        $this->validate($request, [
            'header' => 'required|string|max:255',
            'body' => 'required|string|max:255',
        ]);

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        if ($data->company_id != $company->id) {
            return response()->json([
                'code' => 404,
                'message' => 'FAQ was not found in this company'
            ], 404);
        }

        $data->update($request->all());

        $success['message'] = 'Faq updated';
        $success['code'] =  200;

        return response()->json([
            'faq' => $data,
            'success' => $success,
        ], 200);
    }
```

Gambar 4. 63 *Code* FAQ *Update*

```php
public function destroy(Request $request, $uuid)
    {
        $user = Auth::user();

        $data = Faq::where('uuid', $uuid)->firstOrFail();

        $company = Companies::where('id', $user->company_id)->firstOrFail();

        if ($data->company_id != $company->id) {
            return response()->json([
                'code' => 404,
                'message' => 'FAQ was not found in this company'
            ], 404);
        }

        $data->delete();

        $success['message'] = 'Faq deleted';
        $success['code'] =  200;

        return response()->json([
          'success' => $success,
        ], 200);
    }
```

Gambar 4. 64 *Code* FAQ *Delete*

4.12.7 *Dashboard*

Pada bagian *Dashboard* sendiri terdapat satu fungsi yang ditunjukan pada gambar dibawah. Gambar 4.65 menjelaskan fungsi *show* data *Dashboard* yang bertujuan untuk menampilkan data *Ticket Chat* yang ada pada *database*. Data *Ticket Chat* yang diambil berdasarkan rentan waktu tertentu yang nantinya akan diubah menjadi grafik pada tampilan pengguna.

```php
public function data(Request $request) {
    $user = Auth::user();
    $company = Companies::where('id', $user->company_id)->first();

    $firstDate = Carbon::parse($request->firstDate);
    $lastDate = Carbon::parse($request->lastDate);

    $hour = array();
    $ticketNew = array();
    $ticketDone = array();

    for ($i = 0; $i < 24; $i++) {
        array_push($hour, date('H:i', strtotime($i.':00')));

        $ticketNewCount = Ticket::whereDate('created_at', '>=', $firstDate->isoFormat('YYYY-MM-DD'))
                                ->whereDate('created_at', '<=', $lastDate->isoFormat('YYYY-MM-DD'))
                                ->whereRaw('HOUR(created_at) = '.$i)
                                ->where('company_uuid', $company->uuid)
                                ->count();
        array_push($ticketNew, $ticketNewCount);

        $ticketDoneCount = Ticket::whereDate('close_date', '>=', $firstDate->isoFormat('YYYY-MM-DD'))
                                 ->whereDate('close_date', '<=', $lastDate->isoFormat('YYYY-MM-DD'))
                                 ->whereRaw('HOUR(created_at) = '.$i)
                                 ->where('company_uuid', $company->uuid)
                                 ->count();
        array_push($ticketDone, $ticketDoneCount);
    }

    $response = array(
        'hour' => $hour,
        'ticketNew' => $ticketNew,
        'ticketDone' => $ticketDone,
    );

    return response()->json([
        'code' => 200,
        'data' => $response
    ], 200);
}
```

Gambar 4. 65 *Code Dashboard*

4.12.8 *Ticket Chat*

Pada bagian *Ticket Chat* terdapat tiga fungsi yang ditunjukan pada setiap gambar dibawah. Gambar 4.66 menjelaskan fungsi assign *Ticket Chat* yang bertujuan agar pengguna dari pihak *Company* dapat mengambil alih *Ticket Chat* tersebut. Pengambil alihan *Ticket Chat* ini nantinya akan merubah status dari *Ticket Chat*. Gambar 4.67 menjelaskan fungsi close *Ticket Chat* yang bertujuan untuk menutup *Ticket Chat* tersebut dan mengubah statusnya pada *database*. Gambar 4.68 menjelaskan fungsi assign *Ticket Chat* to yang bertujuan untuk memindahkan

pengguna yang mengambil alih ticket tersebut ke pengguna lain di dalam perusahaan tersebut.

```php
public function assign(Request $request)
    {
        $client = new Client();
        $user = Auth::user();

        if (!$request->message) {
            return response()->json([
                'code' => 400,
                'message' => 'Message required.'
            ], 400);
        }

        if (!$request->ticket) {
            return response()->json([
                'code' => 400,
                'message' => 'Ticket required.'
            ], 400);
        }

        $ticket = Ticket::where('number', $request->ticket)->first();
        if (!$ticket) {
            return response()->json([
                'code' => 404,
                'message' => 'Ticket not found.'
            ], 404);
        }

        $company = Companies::where('id', $user->company_id)->first();
        if (!$company) {
            return response()->json([
                'code' => 404,
                'message' => 'Company not found.'
            ], 404);
        }

        // SEND TO API KC
        try {
            $response = $client->request('PATCH', config('api.url').'/company/'.$company->uuid.'/ticket/'.$request->ticket, [
                'headers' => [
                    'Authorization' => $company->token,
                    'sender' => $user->uuid
                ],
                'json' => [
                    'message' => $request->message
                ]
            ]);

            $ticket->user_id = $user->id;
            $ticket->open_date = date('Y-m-d H:i:s');
            $ticket->open_by = $user->id;
            $ticket->save();

            $success['message'] = 'Data saved.';
            $success['code'] =  200;

            return response()->json([
                'ticket' => $ticket,
                'success' => $success,
            ], 200);

        } catch (ClientException $e) {
            return response()->json([
                'code' => 400,
                'message' => Psr7\str($e->getResponse())
            ], 400);
        }

    }
```

Gambar 4. 66 *Code Ticket Chat Assign*

```
public function close(Request $request)
    {
        $client = new Client();
        $user = Auth::user();

        $ticket = Ticket::where('number', $request->ticket)->first();

        if (!$ticket) {
            return response()->json([
                'code' => 404,
                'message' => 'Ticket not found.'
            ], 404);
        }

        $company = Companies::where('id', $user->company_id)->first();
        if (!$company) {
            return response()->json([
                'code' => 404,
                'message' => 'Company not found.'
            ], 404);
        }

        // SEND TO API KC
        try {

            $response = $client->request('DELETE', config('api.url').'/company/'.$company-
>uuid.'/ticket/'.$request->ticket, [
                'headers' => [
                    'Authorization' => $company->token,
                    'sender' => $user->uuid
                ]
            ]);

            $ticket->status = 0;
            $ticket->close_date = date('Y-m-d H:i:s');
            $ticket->close_by = 'Customer Service';
            $ticket->save();

            $success['message'] = 'Data saved.';
            $success['code'] =  200;

            return response()->json([
                'ticket' => $ticket,
                'success' => $success,
            ], 200);

        } catch (ClientException $e) {
            return response()->json([
                'code' => 400,
                'message' => Psr7\str($e->getResponse())
            ], 400);
        }

    }
```

Gambar 4. 67 *Code Ticket Chat Close*

```php
public function assignTo(Request $request)
    {
        $client = new Client();
        $user = Auth::user();

        if (!$request->message) {
            return response()->json([
                'code' => 400,
                'message' => 'Message required.'
            ], 400);
        }

        if (!$request->byeMessage) {
            return response()->json([
                'code' => 400,
                'message' => 'Bye Message required.'
            ], 400);
        }

        if (!$request->customer_service_id) {
            return response()->json([
                'code' => 400,
                'message' => 'Customer Service Id required.'
            ], 400);
        }

        $ticket = Ticket::where('number', $request->ticket)->first();
        if (!$ticket) {
            return response()->json([
                'code' => 404,
                'message' => 'Ticket not found.'
            ], 404);
        }

        $cs = User::where('id', $request->customer_service_id)->first();
        if (!$cs) {
            return response()->json([
                'code' => 404,
                'message' => 'Customer Service not found.'
            ], 404);
        }

        $company = Companies::where('id', $cs->company_id)->first();
        if (!$company) {
            return response()->json([
                'code' => 404,
                'message' => 'Company not found.'
            ], 404);
        }

        // SEND TO API KC
        try {

            $response = $client->request('PATCH', config('api.url').'/company/'.$company->uuid.'/ticket/'.$request->ticket, [
                'headers' => [
                    'Authorization' => $company->token,
                    'sender' => $cs->uuid
                ],
                'json' => [
                    'message' => $request->message,
                    'byeMessage' => $request->byeMessage
                ]
            ]);

            $ticket->user_id = $request->customer_service_id;
            $ticket->open_date = date('Y-m-d H:i:s');
            $ticket->save();

            $success['message'] = 'Data saved.';
            $success['code'] =  200;

            return response()->json([
                'ticket' => $ticket,
                'success' => $success,
            ], 200);

        } catch (ClientException $e) {
            return response()->json([
                'code' => 400,
                'message' => Psr7\str($e->getResponse())
            ], 400);
        }
    }
```

Gambar 4. 68 *Code Ticket Chat Assign To*

BAB V

PENUTUP

5.1 Kesimpulan

Kesimpulan yang diperoleh dari perancangan sistem aplikasi KLIK Chat Bisnis pada PT. Surya Lifetime International adalah sebagai berikut :

1. Berdasarkan permintaan penyelia untuk membuat sebuah sistem yang dapat mengintegrasikan antara *database*, KLIK Chat Bisnis berbasis *web* dan juga KLIK Chat Bisnis berbasis *mobile*. Maka sistem yang dapat mengintegrasikan ketiganya berhasil dibuat dan telah diuji melalui aplikasi KLIK Chat Bisnis dengan *operating system* android.
2. Sistem aplikasi yang dibuat telah dapat mengolah data-data yang dibutuhkan sesuai dengan proses bisnis pada aplikasi KLIK Chat Bisnis. Maka dapat disimpulkan sistem telah memenuhi kebutuhan dari penyelia dan bisa masuk ke *production stage*.
3. Sistem aplikasi telah menerapkan *framework* Laravel yang bertujuan agar dapat menyesuaikan dengan sistem yang digunakan pada aplikasi KLIK Chat baik berbasis *web* maupun *mobile*. Selain menyesuaikan dengan sistem aplikasi yang sudah ada tujuan penggunaan *framework* ini ialah agar dapat mudah untuk diubah oleh pihak perusahaan tanpa perlu mempelajari *framework* baru.

5.2 Saran

Berdasarkan sistem aplikasi KLIK Chat Bisnis berbasis *mobile* pada PT. Surya Lifetime International maka dapat diberikan beberapa saran sebagai berikut:

1. Diharapkan dikemudian hari sistem aplikasi KLIK Chat Bisnis yang telah dibuat dapat dikembangkan kembali dari sisi penataan *output* (berupa *json*) agar hanya *output* yang penting saja yang diperlihatkan agar dari perangkat android dapat mengolah data yang masuk dengan baik.
2. Diharapkan dikemudian hari sistem aplikasi KLIK Chat Bisnis yang telah dibuat dapat dikembangkan juga dari sisi pengaturan *parameter* pada setiap API nya dan juga pengaturan untuk *pagination* – nya sistem dapat bekerja lebih efisien.
3. Diharapkan dikemudian hari, pihak KLIK Chat dapat melakukan *maintance* untuk menjaga performa dan keamanan tetap baik.

DAFTAR PUSTAKA

Firman, A., Wowor, H., & Najoan, X. (2016). Sistem Informasi Perpustakaan Online Berbasis *Web*. *E-journal Teknik Elektro dan Komputer vol.5 no.2*, 30.

Hidayat, R. (2010). *Cara Praktis Membangun Website Gratis.* Jakarta: Elex Media Komputindo.

KLIKChat. (2018). Retrieved from klikchat.com: https://klikchat.com/#

McCool, S. (2012). *Laravel Starter.* Birmingham: Packt *Publish*ing Ltd.

Otwell, T. (2020). *Laravel Philosophy*. Retrieved from Laravel.com: https://laravel.com/docs/4.2/introduction

Rahman, M. A. (2013). Perancangan dan Implementasi RESTful *Web* Service untuk Game Sosial Food Merchant Saga pada Perangkat Android. *JURNAL TEKNIK POMITS Vol. 2*, 2 - 4.

Rangta, K. (2019). *Learn Testing in 1 Day: Definitive Guide to Learn Software Testing for Beginners.* Guru99.

Setiabudi, D. (2015). Aplikasi E-Commerce Dengan Menggunakan MySQL dan PHP 4. *JURNAL INFORMATIKA Vol. 3, No. 2*, 88-95.