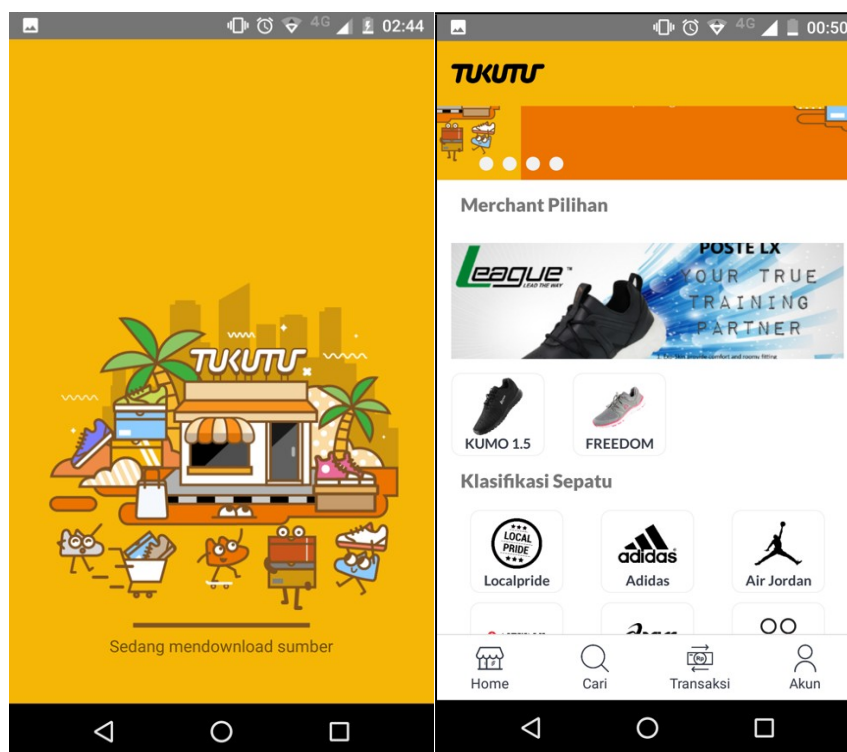


## BAB IV IMPLEMENTASI DAN PEMBAHASAN

### 1.1 Gambaran Umum Sistem

#### 1.1.1 Tukutu

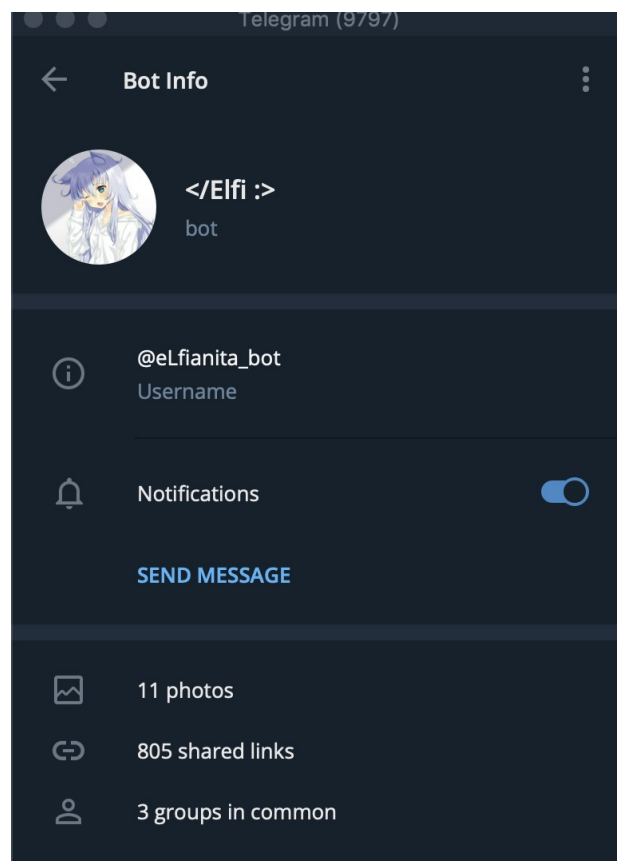


*Gambar 1: Aplikasi Tukutu*

PT Dian Nuswantoro Teknologi adalah perusahaan pengembangan perangkat lunak berbasis web maupun mobile, Aplikasi Tukutu dikembangkan di PT Dinustek. Tukutu merupakan aplikasi yang digunakan untuk jual beli sepatu secara online dengan target merket yaitu pecinta sepatu, yang ingin membeli atau mungkin menjual sepatu koleksi mereka ke masyarakat umum.

Tujuan pengembangan Aplikasi Tukutu lebih untuk membantu para pemilik merk sepatu lokal untuk dapat memasarkan produk mereka sehingga diharapkan dapat bersaing dengan merk internasional yang sudah lebih dahulu dikenal oleh masyarakat luas. Platform Mobile dipilih karena sebagian besar target market place pengguna merupakan pengguna Aplikasi smartphone.

### 1.1.2 Telegram Bot Tukutu



*Gambar 2: Tentang Telegram Bot*

Telegram Bot Tukutu adalah Bot Telegram yang dikembangkan di PT Dinustek yang digunakan untuk melakukan remote terhadap server tukutu, dan juga sebagai alat untuk melakukan integrasi kelanjutan dan pemasangan. selain itu

bot dapat mentranslasikan sebuah text menjadi command line pada server apa bila terjadi hal yang tidak di inginkan dalam keadaan darurat Bot dapat menjadi jalan pertama dalam penanganan server. Fokus pada Bot ini sendiri adalah untuk menjalankan Update pada Microservice yang terintegrasi dengan Git.

Telegram Bot Tukutu dikembangkan dengan teknologi NodeJS yang di kombinasi dengan Shell Script, pesan yang dikirim dari Telegram Bot Tukutu di translasikan menjadi sebuah command line, pada Telegram Bot Tukutu terdapat 2 jenis perintah, yaitu pertama perintah langsung (terdefinisi) merupakan perintah yang sudah diatur menjadi perintah default dan menjadi perintah sekali jalan, kemudian yang kedua adalah perintah kostum (custom) merupakan perintah yang diciptakan dari kreasi System Administrator itu sendiri. Kedua perintah tersebut berawal dari sebuah pesan string yang dikirim dari Telegram Bot Tukutu kemudian di pecah dari perintah utama dan perintah parameter, kemudian text atau String yang sudah tertrlansasi menjadi sebuah perintah Command Line akan di jalankan dalam interpreter Shell yang akan mengeluarkan Stdout dan Stderr, Stdout merupakan keluaran dari perintah yang dijalankan dan Stderr merupakan keluaran berupa pesan error apabila terjadi sebuah error dari perintah yang dijalankan.

## **1.2 Work Flow**

Penggambaran Work Flow Telegram Bot Tukutu seperti pada gambar no. [no\_gambar], Bot berjalan pada latar belakang yang melakukan long polling dengan menunggu request dari pengguna Bot. Request dari pengguna yang dilewatkan sebuah API (Application Programming Interface) dari Telegram Bot berupa sebuah pesan text atau String. Bot membaca text kemudian dilakukan sebuah Parsing Text pada proses belakang, memecah text menjadi beberapa

bagian kemudian melakukan eksekusi sebuah perintah dari pesan text yang di kirim oleh pengguna.

#### 1.2.1 Identifikasi Aktor

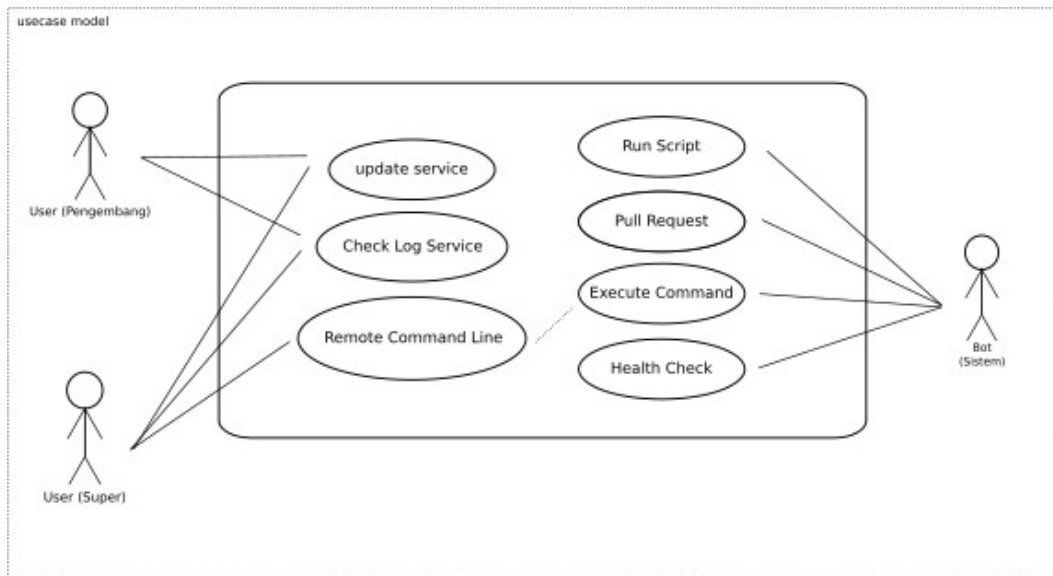
Aktor pada Telegram Bot Tukutu terdiri atas tiga aktor yaitu User(pengembang), User(Super), dan Bot(Sistem). Berikut ini merupakan penjelasan dari tiap aktor yang terlibat.

*Tabel 1: Identifikasi Aktor*

<b>Aktor</b>	<b>Deskripsi</b>
User (Pengembang)	merupakan aktor yang dapat melakukan pengecekan pada microservice yang berjalan pada server tukutu dan dapat melakukan update pada tiap - tiap service yang sedang berjalan
Bot (Sistem)	merupakan aktor yang berperan dalam sistem, dari melakukan pengecekan, pembaruan, serta sistem kerja
User (Super)	merupakan user yang memiliki kendali penuh terhadap Bot itu sendiri, User (Super) dapat melakukan pembaruan terhadap service, melakukan pengecekan, memasukan perintah langsung ke dalam server dengan sebuah pesan yang dapat diterjemahkan menjadi sebuah perintah

## 1.2.2 Use Case Diagram

### 1.2.2.1 Use Case



*Gambar 3: Diagram Use Case*

Dari use case diagram diatas, maka dapat dijelaskan terdapat beberapa aktor dimana aktor tersebut memiliki peran masing - masing. Pada aktor User (Pengembang) dapat melakukan pengecekan pada microservice yang sedang berjalan dan melakukan pembaruan pada service. Lalu pada aktor User (Super) merupakan aktor yang memiliki kendali penuh terhadap Bot, User (Super) akan mendapatkan notifikasi secara pribadi dari Bot jika terdapat pengguna yang tidak terdaftar mencoba menggunakan Bot, User (Super) juga dapat melakukan perintah seperti pada User (Pengembang) dan sebagai tambahan User (Super) dapat melakukan perintah langsung pada Bot yang perintah tersebut akan di eksekusi pada server. Sedangkan Bot (Sistem) memiliki peran dalam proses latar belakang dari input User (Pengembang) dan juga User (Super), seperti proses dalam melakukan pembaruan microservice dan pengechekan service yang sedang

berjalan, mentranslasikan perintah dari User (Super) menjadi sebuah command line yang dapat di eksekusi ke server.

#### 1.2.2.2 Use Case Naratif

Use case naratif digunakan sebagai dokumentasi untuk menjelaskan langkah-langkah yang terjadi pada setiap interaksi dari use case yang telah dibuat. Berikut adalah use case naratif:

<b>Nama Use Case</b>	<b>Aktor</b>	<b>Tujuan</b>	<b>Deskripsi</b>
Update Service	User (Pengembang)	Pembaruan Service	User (Pengembang) dapat melakukan pembaruan sistem pada sebuah service yang berjalan diserver Tukutu
Check Log Service	User (Pengembang)	Pengecekan commit Log	User (Pengembang) dapat melakukan pengecekan commit log yang terdapat pada repository Git
Update Service	User (Super)	Pembaruan Service	User (Super) dapat melakukan pembaruan sistem pada sebuah service yang berjalan diserver Tukutu
Check Log Service	User (Super)	Pengecekan commit Log	User (Super) dapat melakukan

			pengecekan commit log yang terdapat pada repository Git
Remote Command Line	User (Super)	Eksekusi Perintah ke Server	User (Super) dapat melakukan remote command pada server menggunakan Bot, Bot akan mentranslasikan pesan dari User (Super) ke dalam Shell Script
Script Runner	Bot (Sistem)	Eksekusi Shell Script	Bot dapat mengeksekusi Shell Script yang telah dikembangkan oleh Administrator
Pulling Repo	Bot (Sistem)	Pembaruan Source Code	Bot dapat melakukan pembaruan Source Code pada service
Message Translator	Bot (Sistem)	Translasi Pesan ke Shell Command	Bot (Sistem) dapat menerjemahkan pesan menjadi sebuah perintah shell pada server dengan syarat User menambahkan symbol "/" pada pesan yang dikirim di awal kalimat. dan juga perintah

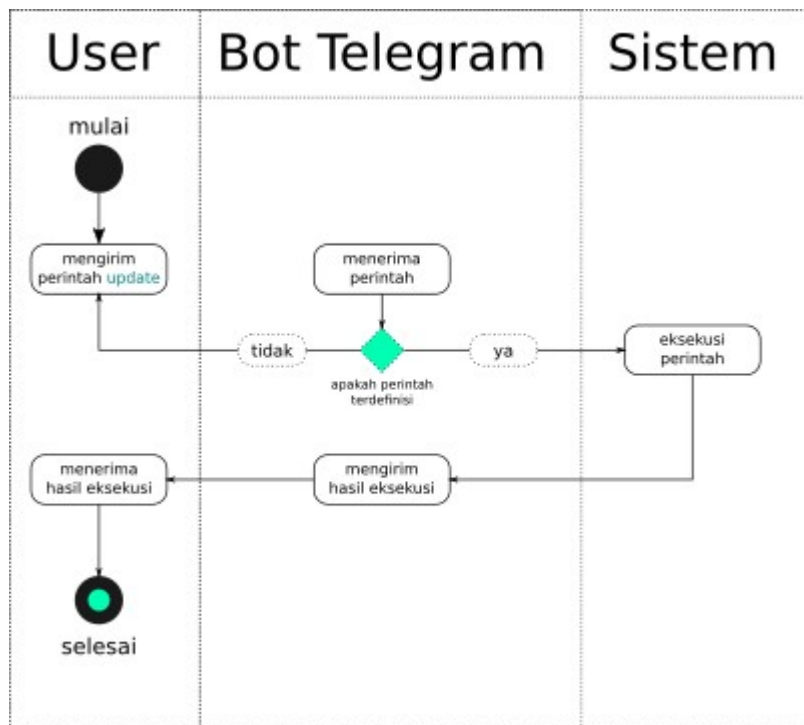
			tersebut harus tersedia pada server.
Healt Check	Bot (Sistem)	Pengechekan Bot	Bot (Sistem) dapat melakukan pengechekan terhadap dirinya sendiri, apabila terjadi sebuah kegagalan sistem maka Bot (Sistem) akan otomatis mengulang kembali proses latar belakang pada dirinya sendiri.

### 1.2.3 Activity Diagram

Pada Tahapan ini akan dijelaskan Alur Activity Diagram pada Telegram Bot Tukutu dari proses Pengguna sampai pada Proses Latar Belakang Bot.



### 1.2.3.1 Activity Diagram Pembaruan Service

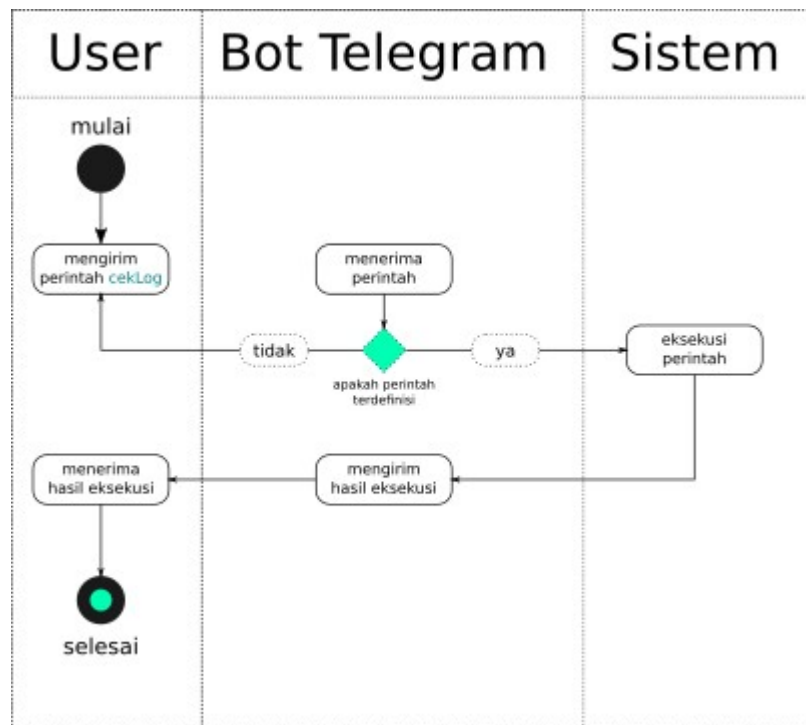


Gambar 4: Flow Diagram Update Service

Pada proses Pembaruan Service yaitu User mengirimkan sebuah pesan Text kepada Telegram Bot Tukutu, pengembang hanya memiliki akses pada perintah yang sudah tersedia pada Telegram Bot Tukutu. pengembang hanya mengirimkan perintah seperti `"/update_service"` kemudian bot akan melakukan eksekusi pada sebuah shell script yang sudah di buat, shell script tersebut memiliki bebrapa inti perintah, yaitu :

- pembaruan commit log.
- pembaruan package atau modul jika ditemukan package atau modul baru.
- migrasi table jika terdeteksi migrasi table baru pada database.
- pembersihan konfigurasi cache pada project.
- pemuatan ulang service pada kontainer.

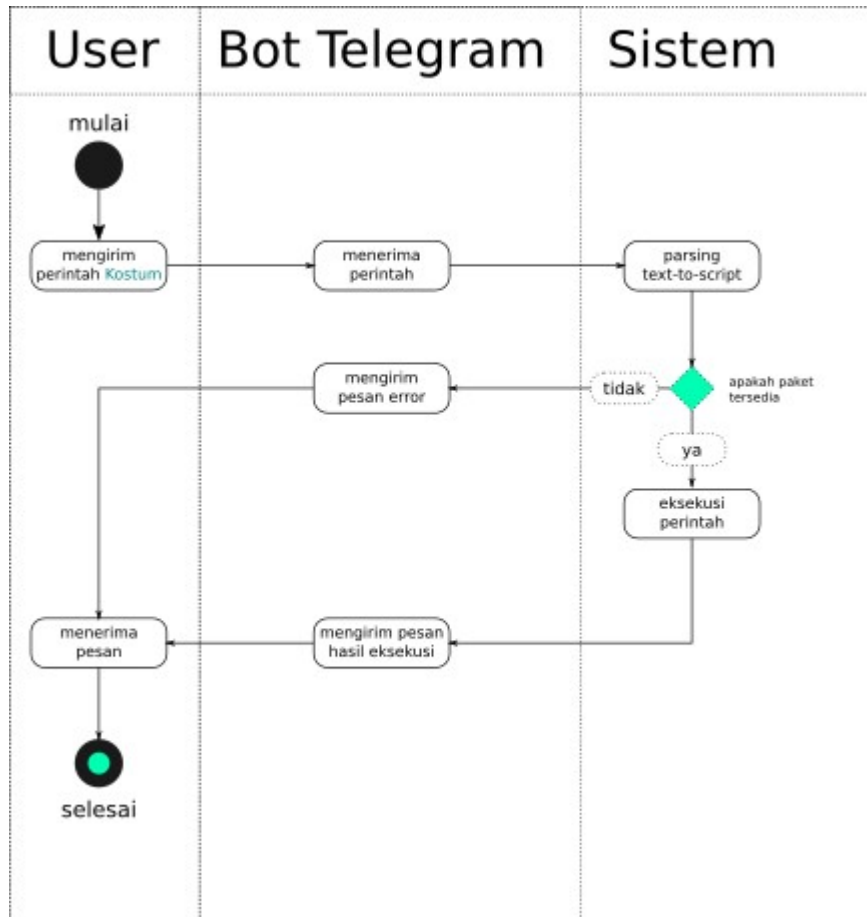
### 1.2.3.2 Activity Diagram Pengecekan Log Service



Gambar 5: Flow Diagram Pengecekan Log Service

Proses pengecekan Log Service diatas User mengirimkan sebuah perintah `/log_service` kepada Telegram Bot Tukutu, proses ini memiliki kesamaan dengan Pembaruan Service, namun pada Proses ini Bot hanya melakukan pengecekan pada commit log repository Git pada service yang ingin di lihat. Bot melakukan eksekusi Shell Script yang sudah ada dan Script memiliki satu inti perintah yaitu pengecekan commit log pada repository.

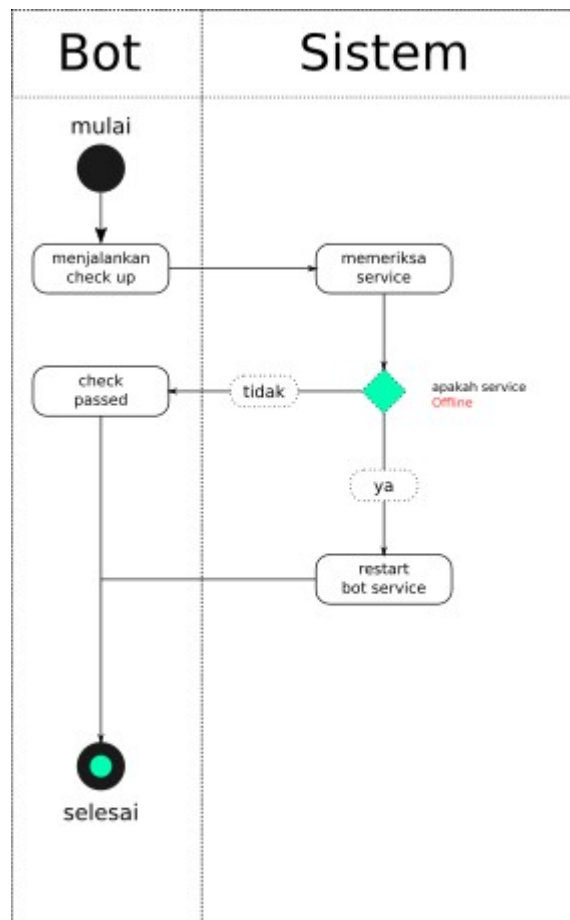
## 1.2.3.3 Activity Diagram Remote Command Line



Gambar 6: Flow Diagram Remote Command Line

Proses remote Command Line merupakan sebuah fitur dari Telegram Bot Tukutu yang tersedia untuk melakukan eksekusi perintah langsung pada server dengan mengirimkan sebuah perintah dengan diawali symbol `"/perintah"` dimana perintah tersebut merupakan binary file atau sebuah package yang sudah tersedia pada server.

## 1.2.3.4 Activity Diagram Health Check

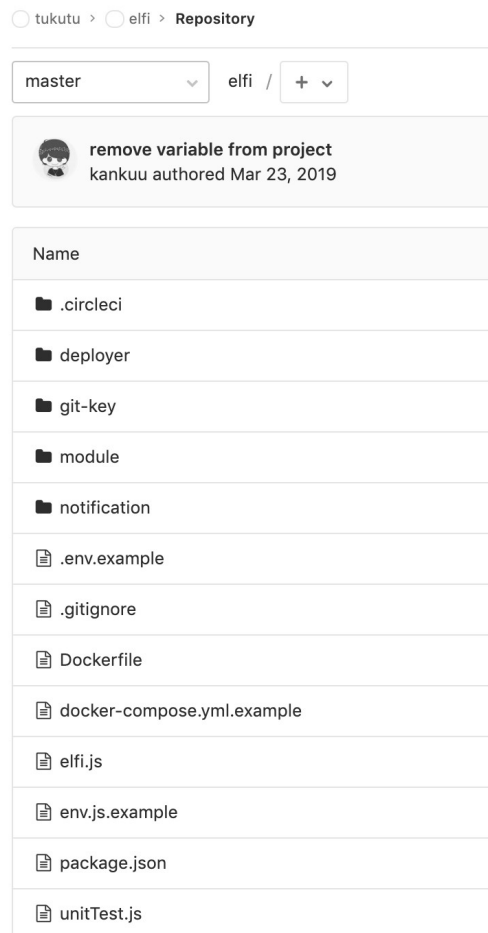


Gambar 7: Flow Diagram Health Check

Health Check berjalan pada latar belakang yang dilakukan penjadwalan setiap satu jam sekali, terdapat sebuah Shell Script yang akan dijalankan untuk melakukan pengecekan pada service Telegram Bot Tukutu yang berjalan di latar belakang. script ini hanya memiliki 2 kondisi yaitu online dan offline. jika service Telegram Bot Tukutu terdeteksi pada posisi Online maka tidak ada tindakan yang akan dilakukan, jika posisi service Telegram Bot Tukutu terdeteksi dalam kondisi Offline maka script akan melakukan pengulangan kembali (restarting) service Telegram Bot Tukutu.

## 1.3 Implementasi Kode Program

### 1.3.1 Struktur Project Telegram Bot Tukutu



*Gambar 8: Struktur Project*

Gambar diatas merupakan struktur project dari Telegram Bot Tukutu, setiap folder diatas mendefinisikan sebuah paket dan memiliki peran masing - masing, berikut deskripsi peran terpenting:

#### 1.3.1.1 .circleci

.circleci memiliki peran dalam continuous integration dari Telegram Bot Tukutu sendiri.

### 1.3.1.2 Deployer

terdapat 5 (lima) file pada paket ini yaitu :

- .gitignore
- lib.sh
- sandbox.sh
- updater.sh
- variable.sh.example

```

lib.sh 2.33 KB
1  #!/usr/bin/env bash
2  # this library of bash function representation of deployer
3  # kind of function is to pull repo, update dependency, and
4  # configuration set of project
5  #
6  # maintainable by <abas> akhmadbasir5@gmail.com
7
8  set -e
9
10 # function to update project
11 function update_repo () {
12     # $git_dir=$1 # git directory
13     # $git_work_tree=$2 # git work tree
14     # $remote=$3 # remote repositories
15     # $branch=$4 # branch to to update
16
17     echo "updating repo.."
18     GIT_PULL=$(git --git-dir=$1 --work-tree=$2 pull $3 $4 > /dev/null 2>&1 ; echo $? )
19     if [[ $GIT_PULL -gt 0 ]]; then
20         echo "error pulling resource, errcode : $GIT_PULL"
21         echo "EOP"
22     else
23         echo "source updated!"
24     fi
25 }
26
27 # function to install dependency with composer
28 function composer_install () {
29     # $service_path=$1 # the working dir
30
31     COMPOSER_INSTALL=$(composer --working-dir=$1 install --no-dev > /dev/null 2>&1 ; echo $? )
32     if [[ $COMPOSER_INSTALL -gt 0 ]]; then
33         echo "composing failed, errcode : $COMPOSER_INSTALL"
34         echo "EOP"
35     else
36         echo "dependency updated"
37     fi
38 }

```

Gambar 9: Shell Script Lib.sh I

```

39
40 # database migration : update table, using docker instead
41 function artisan_migrate_force () {
42     # $service_name=$1 # name of container service
43
44     docker exec -i $1 php artisan migrate --force
45     if [[ $(docker exec -i $1 php artisan migrate --force > /dev/null 2>&1 ; echo $? ) -gt 0 ]];then
46         echo "something wrong here"
47     else
48         echo "table migrated!"
49     fi
50 }
51
52 # dockerizer config restart queue and config clear
53 function artisan_config_cache () {
54     # $service_name=$1 # name of service or container name
55
56     docker exec -i $1 php artisan config:cache
57 }
58 function artisan_config_clear () {
59     # $service_name=$1 # name of service or container name
60
61     docker exec -i $1 php artisan config:clear
62 }
63 function artisan_queue_restart () {
64     # $service_name=$1 # name of service or container name
65
66     docker exec -i $1 php artisan queue:restart
67 }
68
69 # restarting docker container service
70 function docker_compose_restart () {
71     # $service_name=$1 # name of service or container name
72     # $service_path=$2 # path project
73
74     RESTART_CONTAINER=$(docker-compose -f $2/docker-compose.yml restart > /dev/null 2>&1 ; echo $? )
75     if [[ $RESTART_CONTAINER -gt 0 ]];then
76         echo "something wrong when restarting container $1"
77     else
78         echo "container successfully restarted"
79     fi
80 }

```

*Gambar 10: Shell Script Lib.sh II*

lib.sh berisikan berbagai fungsi yang digunakan sebagai kebutuhan service Telegram Bot Tukutu. lib.sh dapat dikatakan sebagai modul shell Script yang memiliki berbagai fungsi, dimana fungsi - fungsi tersebut yang nanti akan di panggil dalam perintah yang di minta.

```

1  #!/usr/bin/env bash
2  # this library of bash function representation of deployer
3  # kind of function is to pull repo, update dependency, and
4  # configuration set of project
5  #
6  # maintainable by <abas> akhmadbasir5@gmail.com
7
8  set -e
9  source $(find . -name lib.sh)
10 source $(find . -name variable.sh)
11
12 update_option=$1
13 case "$update_option" in
14     "--dashboard" | "-d")
15         # dashboard update statement
16         echo "updating on dashboard service"
17         update_repo $DASHBOARD_SERVICE_PATH/.git $DASHBOARD_SERVICE_PATH $DASHBOARD_REMOTE $DASHBOARD_BRANCH
18         composer_install $DASHBOARD_SERVICE_PATH
19         artisan_config_cache $DASHBOARD_SERVICE_NAME
20         artisan_config_clear $DASHBOARD_SERVICE_NAME
21         docker_compose_restart $DASHBOARD_SERVICE_NAME $DASHBOARD_SERVICE_PATH
22         ;;
23     "--mainservice" | "-ms")
24         # mainservice update statement
25         echo "updating on main service"
26         update_repo $MAIN_SERVICE_PATH/.git $MAIN_SERVICE_PATH $MAIN_REMOTE $MAIN_BRANCH
27         composer_install $MAIN_SERVICE_PATH
28         artisan_migrate_force $MAIN_SERVICE_NAME
29         artisan_config_cache $MAIN_SERVICE_NAME
30         artisan_config_clear $MAIN_SERVICE_NAME
31         artisan_queue_restart $MAIN_SERVICE_NAME
32         docker_compose_restart $MAIN_SERVICE_NAME $MAIN_SERVICE_PATH
33         ;;
34     "--userservice" | "-us")
35         # userservice update statement
36         echo "updating on user service"
37         update_repo $USER_SERVICE_PATH/.git $USER_SERVICE_PATH $USER_REMOTE $USER_BRANCH
38         composer_install $USER_SERVICE_PATH
39         docker_compose_restart $USER_SERVICE_NAME $USER_SERVICE_PATH
40         ;;
41     "--merchantservice" | "-cs")
42         # merchantservice update statement
43         echo "updating on merchant service"
44         update_repo $MERCHANT_SERVICE_PATH/.git $MERCHANT_SERVICE_PATH $MERCHANT_REMOTE $MERCHANT_BRANCH
45         composer_install $MERCHANT_SERVICE_PATH
46         docker_compose_restart $MERCHANT_SERVICE_NAME $MERCHANT_SERVICE_PATH
47         ;;

```

*Gambar 11: Shell Script Updater I*



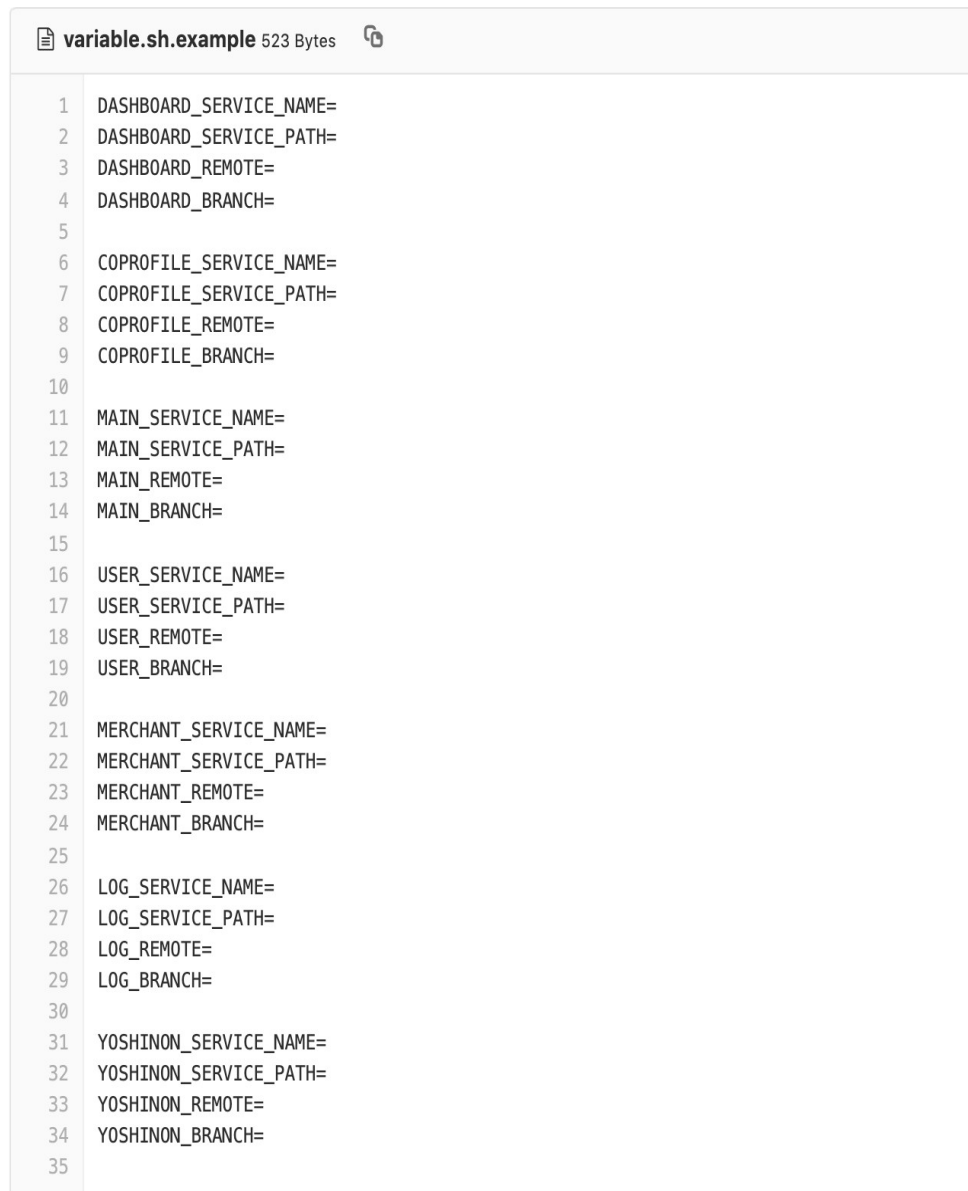
```

48     "--logservice" | "-ls")
49         # logservice update statement
50         echo "updating on log service"
51         update_repo $LOG_SERVICE_PATH/.git $LOG_SERVICE_PATH $LOG_REMOTE $LOG_BRANCH
52         composer_install $LOG_SERVICE_PATH
53         docker_compose_restart $LOG_SERVICE_NAME $LOG_SERVICE_PATH
54     ;;
55     "--coprofile" | "-cp")
56         # company profile update statement
57         echo "updating company profile"
58         update_repo $COPROFILE_SERVICE_PATH/.git $COPROFILE_SERVICE_PATH $COPROFILE_REMOTE $COPROFILE_BRANCH
59         composer_install $COPROFILE_SERVICE_PATH
60         artisan_migrate_force $COPROFILE_SERVICE_NAME
61         artisan_config_cache $COPROFILE_SERVICE_NAME
62         artisan_config_clear $COPROFILE_SERVICE_NAME
63         docker_compose_restart $COPROFILE_SERVICE_NAME $COPROFILE_SERVICE_PATH
64     ;;
65     "--yoshinon" | "-ys")
66         # yoshinon update statement
67         echo "updating yoshinon"
68         update_repo $YOSHINON_SERVICE_PATH/.git $YOSHINON_SERVICE_PATH $YOSHINON_REMOTE $YOSHINON_BRANCH
69     ;;
70 *)
71     echo "nothing option $update_option yet, see the manual bellow"
72     echo "=="
73     echo "\$ updater [option]"
74     echo "=="
75     echo "option:"
76     echo "  --dashboard      -d      update dashboard service"
77     echo "  --mainservice    -ms     update main service"
78     echo "  --userservice    -us     update user service"
79     echo "  --merchantservice -cs     update mserchant service"
80     echo "  --coprofile      -cp     update company profile service"
81     echo "  --yoshinon       -ys     update yoshinon"
82     ;;
83 esac

```

*Gambar 12: Shell Script Updater II*

updater.sh memiliki peran sebagai main script yang akan mengeksekusi perintah berdasarkan parameter yang diberikan, parameter yang diberikan oleh User menentukan perintah apa yang harus dieksekusi, dan updater sh akan memanggil fungsi - fungsi yang ada pada lib.sh dan juga memanggil variable yang tersedia pada variable.sh.

A screenshot of a code editor window. The title bar at the top reads 'variable.sh.example 523 Bytes' with a file icon on the left and a refresh icon on the right. The editor area contains a list of 35 lines of code. Lines 1-5 are for DASHBOARD variables, 6-10 for COPROFILE, 11-15 for MAIN, 16-20 for USER, 21-25 for MERCHANT, 26-30 for LOG, and 31-35 for YOSHINON. Each group consists of four lines: SERVICE\_NAME, SERVICE\_PATH, REMOTE, and BRANCH, all followed by an equals sign.

```
1 DASHBOARD_SERVICE_NAME=  
2 DASHBOARD_SERVICE_PATH=  
3 DASHBOARD_REMOTE=  
4 DASHBOARD_BRANCH=  
5  
6 COPROFILE_SERVICE_NAME=  
7 COPROFILE_SERVICE_PATH=  
8 COPROFILE_REMOTE=  
9 COPROFILE_BRANCH=  
10  
11 MAIN_SERVICE_NAME=  
12 MAIN_SERVICE_PATH=  
13 MAIN_REMOTE=  
14 MAIN_BRANCH=  
15  
16 USER_SERVICE_NAME=  
17 USER_SERVICE_PATH=  
18 USER_REMOTE=  
19 USER_BRANCH=  
20  
21 MERCHANT_SERVICE_NAME=  
22 MERCHANT_SERVICE_PATH=  
23 MERCHANT_REMOTE=  
24 MERCHANT_BRANCH=  
25  
26 LOG_SERVICE_NAME=  
27 LOG_SERVICE_PATH=  
28 LOG_REMOTE=  
29 LOG_BRANCH=  
30  
31 YOSHINON_SERVICE_NAME=  
32 YOSHINON_SERVICE_PATH=  
33 YOSHINON_REMOTE=  
34 YOSHINON_BRANCH=  
35
```

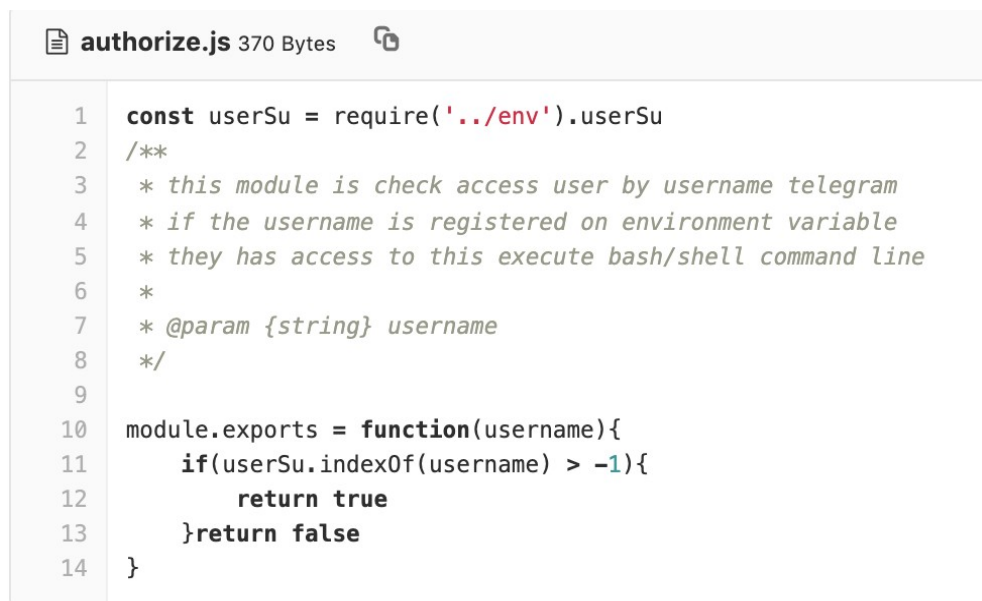
*Gambar 13: Variable.sh*

variable.sh.example merupakan file yang berisikan variable - variable yang disediakan pada tiap service, variable.sh.example merupakan file yang merupakan file yang dinamis harus dipisahkan, variable.sh.example akan di ubah penamaan mennjadi variable.sh dan isi dari tiap - tiap variable harus diisi dengan nilai yang mutlak, karena variable disini berperan penting dalam penentuan jalannya script.

### 1.3.1.3 Module

paket module berisikan modul - modul yang tertulis dengan bahasa pemrograman javascript native, pemisahan module dilakukan untuk kemudahan dalam penggunaan ulang, ada beberapa module terpenting sebagai berikut :

- authorize.js
- git.js
- log.js
- splitter.js



The screenshot shows a code editor window titled 'authorize.js 370 Bytes'. The code is as follows:

```

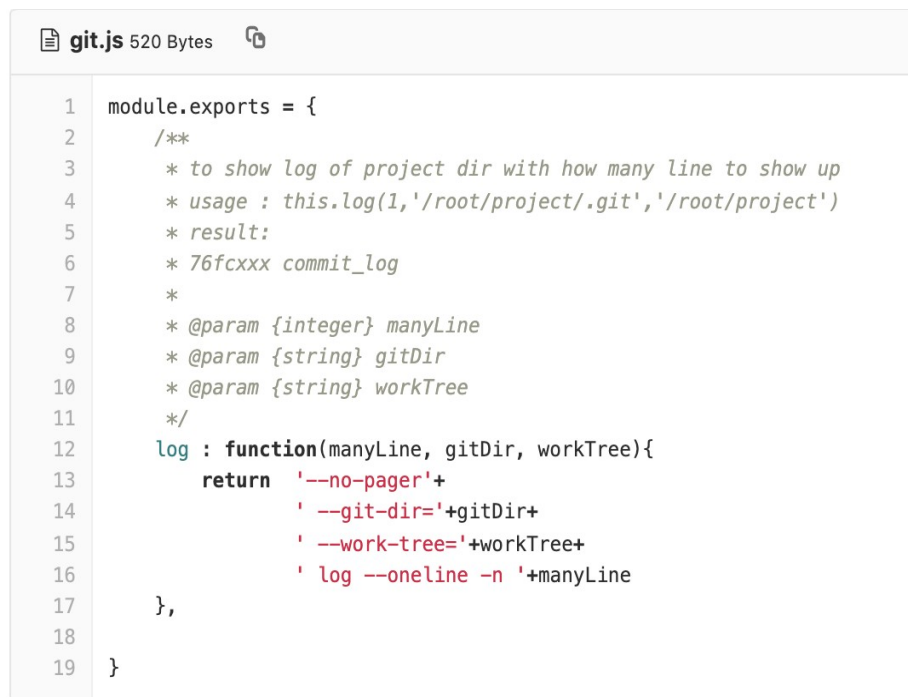
1  const userSu = require('../env').userSu
2  /**
3   * this module is check access user by username telegram
4   * if the username is registered on environment variable
5   * they has access to this execute bash/shell command line
6   *
7   * @param {string} username
8   */
9
10 module.exports = function(username){
11     if(userSu.indexOf(username) > -1){
12         return true
13     }return false
14 }

```

Gambar 14: Logika Auththorize Module

module authorize.js berfungsi sebagai pengecekan pengguna Telegram Bot Tukutu, merupakan otentikasi sederhana dengan membandingkan list user yang terdefinisi dengan user yang sedang melakukan chat dengan Bot, jika user terdeteksi tidak dalam list pengguna Telegram Bot Tukutu maka Bot akan tidak melakukan apapun pada pesan user tersebut tapi Bot akan mengirimkan sebuah

pesan notifikasi pada User (Super) bahwa user yang tidak terdaftar tersebut telah melakukan interaksi dengan Bot.



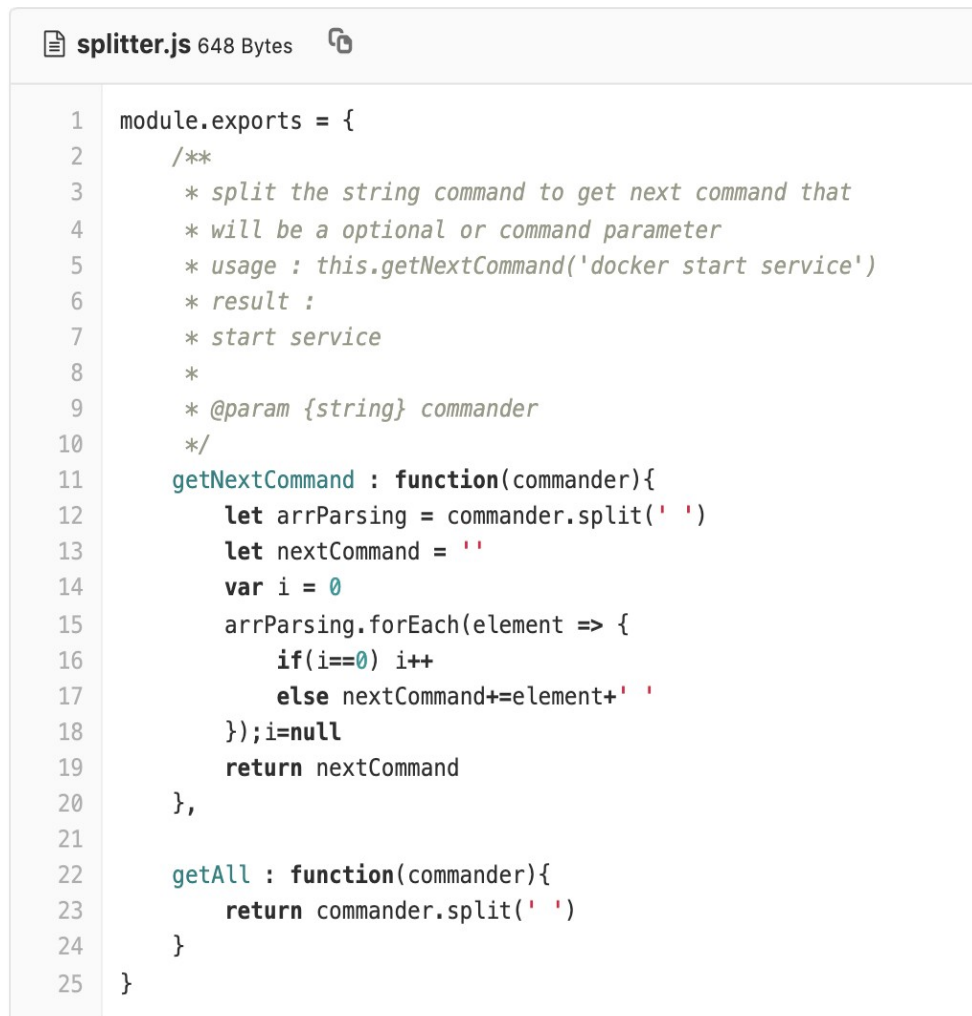
```

1  module.exports = {
2    /**
3     * to show log of project dir with how many line to show up
4     * usage : this.log(1, '/root/project/.git', '/root/project')
5     * result:
6     * 76fcxxx commit_log
7     *
8     * @param {integer} manyLine
9     * @param {string} gitDir
10    * @param {string} workTree
11    */
12    log : function(manyLine, gitDir, workTree){
13      return '--no-pager'+
14        ' --git-dir='+gitDir+
15        ' --work-tree='+workTree+
16        ' log --oneline -n '+manyLine
17    },
18  }
19 }

```

*Gambar 15: Logika Git.js Module*

module git.js berfungsi untuk pengecekan commit log pada git repository local server, untuk memudahkan pada proses pemanggilan git.js menyediakan fungsi dengan sebuah parameter directory yang dituju.



```

1  module.exports = {
2      /**
3       * split the string command to get next command that
4       * will be a optional or command parameter
5       * usage : this.getNextCommand('docker start service')
6       * result :
7       * start service
8       *
9       * @param {string} commander
10      */
11     getNextCommand : function(commander){
12         let arrParsing = commander.split(' ')
13         let nextCommand = ''
14         var i = 0
15         arrParsing.forEach(element => {
16             if(i==0) i++
17             else nextCommand+=element+' '
18         });i=null
19         return nextCommand
20     },
21
22     getAll : function(commander){
23         return commander.split(' ')
24     }
25 }

```

*Gambar 16: Logika Splitter.js Module*

module splitter.js memiliki fungsi sebagai pemecah text atau String, pesan yang dikirim dari pengguna akan dipecah berdasarkan spasi, dan diambil binary package dari kata pertama yang tersedia.

#### 1.3.1.4 Notification

Dalam paket Notification terdapat fungsi untuk melakukan Heath Checking, proses pengecekan pada diri Telegram Bot Tukutu itu sendiri dengan

menggunakan penjadwalan secara berkala yaitu satu jam sekali, berikut merupakan gambaran script health check pada service Telegram Bot Tukutu.



```
1  #!/usr/bin/env bash
2  if [ $(pm2 status | grep stopped | grep elfi > /dev/null 2>&1 ;echo $? ) -eq 0 ];then
3      echo "restarted elfi on $(date)" >> elfi/log/restart.log
4      pm2 restart elfi --watch
5  else
6      echo "healthy check passed!"
7  fi
```

*Gambar 17: Shell Script Restart Bot Service*

#### 1.3.1.5 Elfi.js

Selain 4 (empat) Paket diatas, ada main program utama yang menjalankan semua proses tersebut, yaitu elfi.js. Elfi.js merupakan main program yang berjalan di latar belakang menggunakan proses NodeJS, elfi.js berjalan dengan metode long polling, melakukan listen request pada dengan integrasi Telegram Bot API. elfi.js pada Telegram Bot Tukutu memiliki fungsi utama yaitu :

- execute function

```

34  /**
35   * @param {string} first_command
36   * @param {string} following_command
37   */
38  function execute(cmd, fCmd, chat_id) {
39    let script = cmd + ' ' + fCmd
40    log.resMessage(script)
41    exec(script, (error, stdout, stderr) => {
42      if(stderr.includes('/bin/sh') && stderr.includes('not found')){
43        stderr = "i am sorry, i dont know what to do >.<"
44      }
45      if (error) {
46        // log.resExecErr(error)
47        log.resStdout(stdout)
48        log.resStderr(stderr)
49        if(stderr != null){
50          bot.sendText(chat_id, stderr)
51        }
52        bot.sendText(chat_id, stdout)
53      } else if (stdout == '' || stdout == null) {
54        bot.sendText(
55          chat_id,
56          'command execute with stdout `null` result'
57        )
58        log.resMessage('execute with null output')
59      } else {
60        if(stderr != null){
61          bot.sendText(chat_id, stderr)
62        }
63        bot.sendText(chat_id, stdout)
64        log.resStdout(stdout)
65      }
66    })
67  }

```

*Gambar 18: Logika Execute Elfi.js*

execute function merupakan fungsi untuk mengkonversi pesan dari User, pesan diterjemahkan dalam perintah shell script dan kemudian akan dieksekusi pada server.

pada beberapa kasus dalam perintah server, ada perintah - perintah yang harus dihindari, seperti melakukan perintah poweroff, halt, reboot, etc perintah - perintah yang seharusnya tidak boleh di eksekusi oleh Telegram Bot Tukutu. dalam hal tersebut elfi.js akan melakukan pengecekan pada sebuah kondisi apakah pesan yang berupa perintah ke server bukan merupakan perintah yang sensitif, dijelaskan pada kode berikut.

```
restart.sh 234 Bytes
1  #!/usr/bin/env bash
2  if [ $(pm2 status | grep stopped | grep elfi > /dev/null 2>&1 ;echo $?) -eq 0 ];then
3      echo "restarted elfi on $(date)" >> elfi/log/restart.log
4      pm2 restart elfi --watch
5  else
6      echo "healthy check passed!"
7  fi
```

*Gambar 19: Logika Retristed Command*

- update service

```
case 'main_update':
    execute(bash, update+' --mainservice')
    bot.sendText(chat.id,'updating.... #main_service',{replyToMessageId:messageId})
    bot.sendChatAction(chat.id,'typing',(status)=>{
        if(status.Error){
            bot.sendText(chat.id,"hmm, cant you check my logs? there must be error there.")
        }else{
            setTimeout(()=>{
                bot.sendText(chat.id,'source has been updated, check the last commit',{
                    disable_notification: false,
                    disable_web_page_preview: true,
                    replyToMessageId: messageId
                },()=>{
                    execute('git',git.log(
                        6,env.service.main+'/.git',env.service.main
                    ),chat.id)
                })
            },5000)
        }
    })
    break
```

*Gambar 20: Logika Update Service*



Gambar diatas merupakan salah satu kode yang akan menjalankan perintah pembaruan sistem pada service Tukutu, terjadi beberapa proses pada opsi tersebut, yaitu pembaruan pada latar belakang kemudian setelah pembaruan akan dilakukan pengecekan commit log, dan hasil dari commit log akan di kirimkan kepada user.

- log service

```
case 'glmain':
  bot.sendChatAction(chat.id, 'typing', ()=>{
    bot.sendText(chat.id, "history of commit log", {
      replyToMessageId: messageId
    }, ()=>{
      execute('git', git.log(
        6, env.service.main+'/.git', env.service.main
      ), chat.id)
    })
  })
break
```

*Gambar 21: Logika Pengecekan Log Service*

Gambar diatas merupakan salah satu kode yang akan menjalankan perintah pengecekan commit log pada service Tukutu. Bot akan melakukan pengecekan dari History commit log, di ambil 5 history terakhir yang akan di kirimkan kembali kepada user.

- Unauthorize User

```

} else {
  let unAuthorizeUsername = 'this user : @' + from.username + '\n' +
    'try to accessing bot with bash/shell command line!\n' +
    'follow this link https://t.me/' + from.username +
    ' to contact this user.\n\n'+
    'command : '+command
  if(command === 'start'){
    bot.sendText(chat.id, 'wasup?')
    bot.sendText(env.chat_id, 'user : @'+from.username+' starting me!')
    log.resExec(from.username, command)
  }else{
    /**
     * if user send command that no have access
     */
    bot.sendText(chat.id, ysnRes.noAccess)
    log.resMessage(unAuthorizeUsername)
    // reporting message to env.chat_id : @kankuu telegram user admin
    bot.sendText(env.chat_id, unAuthorizeUsername)
  }
}
return
`

```

*Gambar 22: Logika Unauthorize User Check*

Jika terdapat user yang mencoba mengakses Telegram Bot Tukutu secara private, maka User (Super) akan mendapatkan notifikasi, dari kode diatas, Bot juga akan menulis Log pada sistem.

#### 1.4 Pengujian

Setelah pengkodean sistem dan sistem (dalam hal ini adalah aplikasi) sudah dapat digunakan. Maka sistem/aplikasi tersebut harus diuji terlebih dahulu. Pada penelitian kali ini penulis menggunakan teknik pengujian :

#### 1.4.1 White Box Testing

White Box merupakan metode desain uji kasus yang menggunakan struktur kontrol dari desain prosedural untuk menghasilkan kasus-kasus uji [19]. Menggunakan metode ujicoba ini dapat menghasilkan kasus-kasus uji coba seperti:

1. Menjamin bahwa seluruh independent paths dalam modul telah dilakukan sedikitnya satu kali.
2. Melakukan seluruh keputusan logikal baik dari sisi benar maupun salah.
3. Melakukan seluruh perulangan sesuai batasannya dan dalam batasan operasionalnya.
4. Menguji struktur data internal untuk memastikan validitasnya.

##### 1.4.1.1 Pseudocode

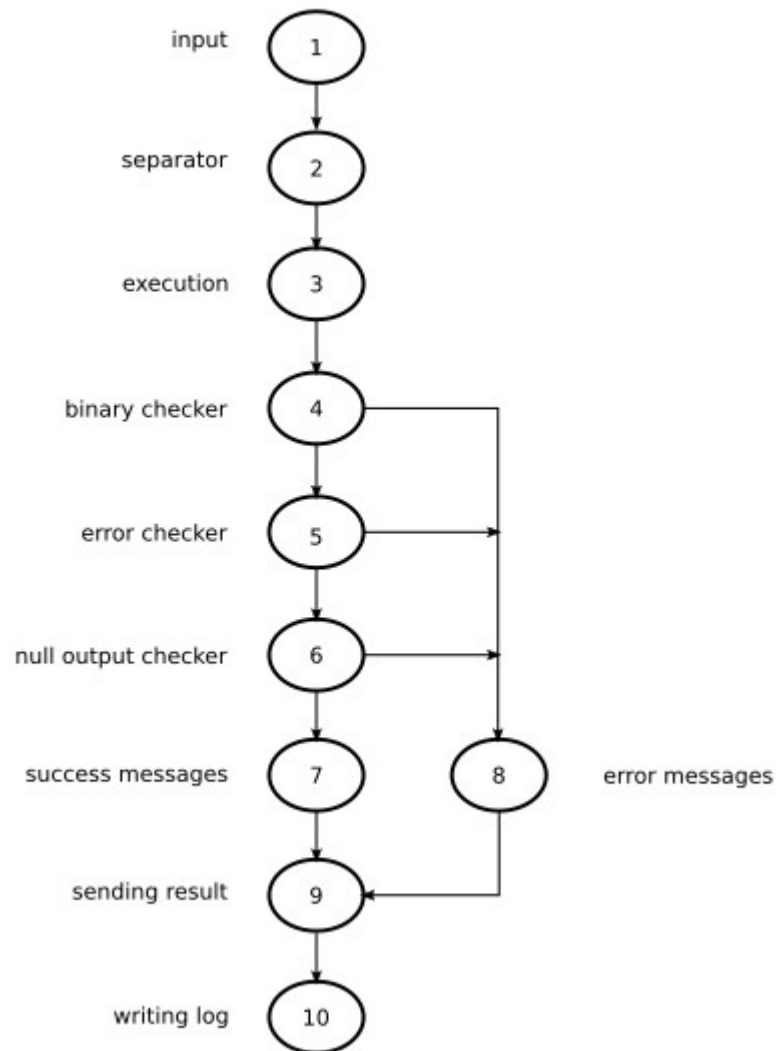
```
function execute (cmd, fCmd, chat_id)
  script <- cmd + ' ' + fCmd
  write log from script
  start exec script begin
    anonim function (error, stdout, stderr)
      if stderr includes /bin/sh, and stderr includes not found then
        stderr <- "i am sorry, i dont know what to do"
      endif

      if error then
        write log from stdout
        write log from stderr
        if stderr != null then
          bot send text notif stderr
        endif
        bot send text stdout

      else if stdout was "", or stdout was null then
        bot send text 'command execute with stdout `null` result'
```

```
write log from 'execute with null output'  
  
else  
  if stderr != null then  
    bot send text stderr  
  endif  
  bot send text stdout  
  write log from stdout  
  
endif  
end anonim function  
end exec script  
end function
```

#### 1.4.2 Pembuatan Flowgraph



Gambar 23. Flowgraph Parsing Algorithm

##### 1.4.2.1 Perhitungan Cyclomatic Complexity

Setelah flowchart dalam bentuk flowgraph, maka langkah selanjutnya adalah menghitung Cyclomatic complexity  $V(G)$  dengan rumus :

$$V(G) = E - N + 2$$

Dimana:

E = jumlah *edge* pada grafik

N = jumlah *node* pada grafik

Dari flowgraph pada gambar 40 dapat dihitung cyclomatic complexitynya sebagai berikut :

$$V(G) = 12 - 10 + 2$$

$$V(G) = 4$$

Jadi dari perhitungan diatas didapatkan bahwa cyclomatic complexity dari flowgraph gambar 4.23 adalah 4.

#### 1.4.2.2 Penentuan jalur independen

Dari perhitungan cyclomatic complexity tadi akan ditentukan jalur independen. Berdasarkan hasil perhitungan maka akan ada 4 jalur independen :

**Jalur 1 = 1-2-3-4-8-9-10**

keterangan : pengguna melakukan input berupa text command, kemudian di pisah menjadi 2 bagian perintah, pertama adalah perintah dasar dari binary file yang akan di jalankan. Jika binary yang di panggil tidak ada maka hasil output akan berupa error, dan bot akan mengirim error kepada pengguna dan diakhir dengan penulisan log error.

**Jalur 2 = 1-2-3-4-5-8-9-10**

keterangan : setelah melewati proses pengecekan binary, proses selanjutnya adalah melakukan pengecekan error execution script. Jika terjadi sebuah error bot akan mengirim error kepada pengguna dan diakhir dengan penulisan log error.

**Jalur 3 = 1-2-3-4-5-6-8-9-10**

keterangan : setelah melewati proses pengecekan error execution script selanjutnya adalah pengecekan null output, karena tidak semua output dari proses dikeluarkan oleh system. Maka jika terjadi null output bot akan mengirim pesan berupa null output kepada pengguna dan diakhir dengan penulisan log error.

**Jalur 4 = 1-2-3-4-5-6-7-9-10**

keterangan : jika input yang diberikan oleh pengguna tidak ada permasalahan maka proses execution script akan dianggap sukses oleh system, dan bot akan mengirim pesan log kepada pengguna dan diakhir dengan penulisan log error.

**1.4.2.3 Hasil Test Case**

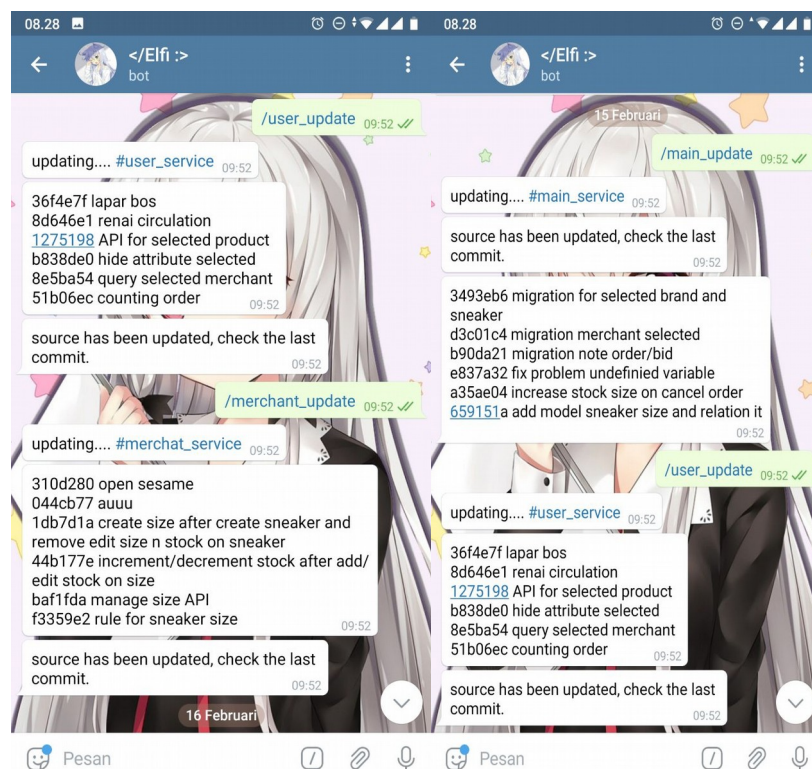
Hasil pengujian test case dapat dilihat pada tabel di bawah ini :

No	Test Case										Ketercapaian	
											Ya	Tidak
1	1	2	3	4	8	9	10				V	

2	1	2	3	4	5	8	9	10		V	
3	1	2	3	4	5	6	8	9	10	V	
4	1	2	3	4	5	6	7	9	10	V	

*Tabel 4.2 Table Hasil Test Case White Box*

#### 1.4.3 Hasil Pengujian Pembaruan Service Tukutu



*Gambar 24: Pengujian Pada Telegram Bot*

Pengujian dilakukan pada Telegram Bot Elfi, response pertama yaitu Bot sedang melakukan Update dan diikuti dengan response commit log lima baris terakhir dari sumber kode.



#### 1.4.4 Hasil Proses Pada Latar Belakang Service

```

/bin/bash /home/sochen/elixir/deployer/updater.sh --main-service
[log write], chat by :cipowela Fri Feb 15 2019 02:52:11 GMT+0000 (UTC) /main_update
git --no-pager --git-dir=/home/dev/dryer/main_service/.git --work-tree=/home/dev/dryer/main_service log --oneline -n 6
stdout :
3493eb6 migration for selected brand and sneaker
d3c01c4 migration merchant selected
b90da21 migration note order/bid
e837a32 fix problem undefined variable
a35ae04 increase stock size on cancel order
659151a add model sneaker size and relation it

stdout :
updating on main service
updating repo..
source updated!
dependency updated
Migrating: 2019_02_11_041834_add_costFee_and_selcected_merchant
Migrated: 2019_02_11_041834_add_costFee_and_selcected_merchant
Migrating: 2019_02_14_064526_add_table_column_selected_sneaker
Migrated: 2019_02_14_064526_add_table_column_selected_sneaker
Migrating: 2019_02_14_064556_add_table_column_selected_sneaker_brand
Migrated: 2019_02_14_064556_add_table_column_selected_sneaker_brand
table migrated!
Configuration cache cleared!
Configuration cached successfully!
Configuration cache cleared!
Broadcasting queue restart signal.
container successfully restarted

```

*Gambar 25: Log Report Pada Pembaruan Service*

Pada gambar diatas merupakan log hasil eksekusi dari perintah pembaruan service tukutu.