2. LANDASAN TEORI

2.1. Proses Bisnis Tour and Travel

Proses transaksi penjualan tiket selama ini dilakukan dengan menggunakan *website*, melalui telepon, ataupun melalui *chat*. Transaksi dilakukan oleh pelanggan yang disebut sebagai *channel* atau *sub agent*. *Channel* ini merupakan reseller yang telah bekerja sama dengan perusahaan.

Untuk melakukan pemesanan tiket, pelanggan dapat langsung menghubungi karyawan perusahaan melalui telepon atau *chat*. Selain itu, pelanggan juga dapat mengakses *website* perusahaan, kemudian melakukan pemesanan dengan cara melakukan pencarian terhadap tiket yang akan dibeli, setelah itu *user* akan memilih tiket yang dicari. Setelah memilih tiket, sistem akan mengecek saldo deposito pelanggan. Jika saldo memenuhi maka pelanggan dapat mengisi data – data yang diperlukan.

2.2. Android Studio

Android Studio merupakan sebuah *software* resmi untuk pengembangan aplikasi yang menggunakan *platform* Android (Developer, Android Studio, 2017). Pembuatan Android Studio diumumkan pada 16 Mei 2013 dan versi stabilnya pertama kali dikeluarkan di akhir tahun 2014. Fitur yang disediakan adalah sebagai berikut:

- *Build* sistem menggunakan *Graddle* yang fleksibel
- Mempunyai emulator yang cepat dan memiliki banyak fitur
- Mempunyai *unified environment* dimana pengguna dapat mengembangkan aplikasi untuk semua *device* Android
- Dapat di *run* secara langsung dengan menambahkan perubahan ke dalam aplikasi yang sedang berjalan tanpa membuat apk yang baru
- Mempunyai *code templates* dan dapat diintegrasikan dengan GitHub untuk membantu pembuatan aplikasi dengan fitur yang sama dan dapat mengimport contoh *code*

- Memiliki banyak alat untuk *testing* dan banyak *framework*
- Memiliki *lint tools* untuk mengetahui performa, kegunaan, perbandingan versi, dan masalah yang lain
- Didukung oleh C++ dan NDK
- Didukung secara langsung oleh Google Cloud Platform

Untuk me-*request web service* melalui Android Studio diperlukan sebuah *class* yang merupakan turunan dari AsyncTask yang sudah ada di dalam *library* Android Studio. Didalam fungsi AsyncTask inilah aplikasi dapat me-*request web service* melalui *object* dari *class* DefaultHttpClient. Aplikasi juga dapat me-*request web service* dengan menyertakan data berbentuk JSON melalui *object* HttpPost seperti yang dijelaskan pada Gambar 2.1, dimana program melakukan *request* dengan menyertakan data nrp dalam bentuk JSON.

```
public void getData(){
    class GetDataJSON extends AsyncTask<String,Void,String> {

        @Override
        protected String doInBackground(String... params) {
            DefaultHttpClient httpclient=new DefaultHttpClient(new BasicHttpParams());
            HttpPost httppost=new HttpPost(URL);

            JSONObject jsonObject = new JSONObject();
            try {
                jsonObject.accumulate("nrp", nrp);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            myJson = jsonObject.toString();
            StringEntity se = null;
            try {
                se = new StringEntity(myJson);
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }

            httppost.setEntity(se);
            httppost.setHeader("Accept", "application/json");
            httppost.setHeader("Content-type","application/json");
            InputStream inputStream=null;
            String result=null;
            try {
                HttpResponse response=httpclient.execute(httppost);
```

Gambar 2.1 Mengirim JSON ke *web service* dengan AsyncTask

Untuk menerima *response* balasan dari *web service* yang telah aplikasi panggil, diperlukan sebuah *object* HttpResponse yang akan menerima *response* dari *web service* dan mengolahnya seperti yang dijelaskan pada Gambar 2.2. Semua proses tersebut di *handle* oleh *class* turunan dari AsyncTask yang telah dibuat. Untuk proses eksekusinya aplikasi dapat membuat *object* dari *class* tersebut dan

kemudian melakukan eksekusi pada *object* tersebut seperti digambarkan pada Gambar 2.3.

```
InputStream inputStream=null;
String result=null;
try {
    HttpResponse response=httpclient.execute(httppost);
    HttpEntity entity=response.getEntity();

    inputStream=entity.getContent();

    BufferedReader reader=new BufferedReader(new InputStreamReader(inputStream,"
        UTF-8"),8);
    StringBuilder sb=new StringBuilder();

    String line=null;
    while ((line=reader.readLine())!=null){
        sb.append(line+"\n");
    }
    result=sb.toString();
}
catch (Exception e){}
finally {
    try {
        if(inputStream!=null)
            inputStream.close();
    }
    catch (Exception squish){}
}
return result;
}
```

Gambar 2.2 Menerima data JSON dari *web service* dengan AsyncTask

```
    protected void onPostExecute(String result){
        if(result != null) {
            myJson = result;
            showList();
        }
    }
}

GetDataJSON g = new GetDataJSON();
g.execute();
}
```

Gambar 2.3 Proses terakhir AsyncTask dan proses eksekusi AsyncTask

2.3. Chatbot

Chatbot atau yang dikenal juga sebagai *bot* atau *chatterbots* atau *conversational agents* merupakan program komputer yang melakukan percakapan dengan manusia via suara ataupun *text*. (Hill, J., 2015). *Design* dari c*hatbot* sudah berkembang dengan sangat canggih. *Chatbot* bahkan digunakan di beberapa sektor bidang seperti sektor edukasi, e-*commerce*, *entertainment*, dan sektor publik. Beberapa *chatbot* menggunakan sistem Natural Language Processing (NLP) yang

canggih, namun ada juga *chatbot* yang menggunakan sistem yang lebih mudah seperti mencari kata kunci dari *input* kemudian menghasilkan balasan dengan kata kunci yang hampir sama atau dengan mencari pola kata yang hampir sama di dalam *database*.

Saat ini *chatbot* sudah mulai berkembang. *Chatbot* saat ini digunakan dalam *virtual assistants* seperti Google Assistant, dan sudah banyak digunakan dalam aplikasi ataupun *website* perusahaan. *Chatbot* juga dimanfaatkan berbagai aplikasi *instant messaging* seperti Facebook Messenger dan Telegram.

Untuk membuat *chatbot* diperlukan sebuah *webhook. Webhook* ini akan dipanggil ketika pengguna *chatbot* melakukan *event-event* yang telah diatur sebelumnya, kemudian *webhook* akan memanggil *web service* yang berkaitan pada *server*. Di dalam *server*, *web service* yang dipanggil akan dijalankan kemudian *webhook* akan melakukan perintah selanjutnya.

Chatbot tour and travel ini dibuat hanya untuk Facebook Messenger menggunakan Application Programming Interface (API) yang telah disediakan oleh Facebook Messenger. Facebook Messenger Bot API ini dapat mengirimkan data ke *webhook* kemudian mengembalikan data dalam bentuk *text*, *button*, *list*, maupun *carousel*. Berikut merupakan *source code* untuk menerima *text* dari Facebook Messenger *chatbot* serta mengirim data ke *chatbot* :

```
import sys, json, traceback, requests
from flask import Flask, request

application = Flask(__name__)
app = application
PAT = 'EAAEiw9ZBVSMoBALPJq1NOmquxXgDBvse7EE9gwl2ZC2EJujo2eCJO9fMcZBz19c1ZBcJMtS3wNlxPrZAJZCqe3PpPX
O2NSGptSRLaXumoN0O401ZCuF1PgFN3jJsZAqMg2spVEgFsxUr1XRfTmHaBm25Pn4A0VZCY5FZCC0mDlVOP0PwZDZD'
VERIFICATION_TOKEN = 'your_own_token'

@app.route('/', methods=['GET'])
def handle_verification():
    print "Handling Verification."
    if request.args.get('hub.verify_token', '') == VERIFICATION_TOKEN:
        print "Webhook verified!"
        return request.args.get('hub.challenge', '')
    else:
        return "Wrong verification token!"
```

Gambar 2.4 Verifikasi token pada *webhook*

Gambar 2.4 memperlihatkan contoh modul untuk verifikasi *token* pada *webhook* pada route '/' dengan metode GET. Modul ini digunakan pada saat pendaftaran *webhook* melalui *website developer* dari Facebook Messenger.

Pendaftaran *webhook* dilakukan dengan mengirim url dimana *webhook* dipasang serta mengirim data verifikasi *token*. Pada saat melakukan pendaftaran *webhook*, Facebook Messenger akan memanggil modul ini kemudian melakukan verifikasi *token* yang dikirim. Jika benar maka *webhook* diterima oleh *chatbot* sehingga semua data yang masuk ke *chatbot* akan diteruskan ke *webhook* tersebut.

```
@app.route('/', methods=['POST'])
def handle_messages():
    payload = request.get_data()

    # Handle messages
    for sender_id, message in messaging_events(payload):
        # Start processing valid requests
        try:
            response = processIncoming(sender_id, message)

            if response is not None:

                if response == 'button':
                    send_button(PAT, sender_id)

                else:
                    send_message(PAT, sender_id, response)

            else:
                send_message(PAT, sender_id, "Sorry I don't understand that")
        except Exception, e:
            print e
            traceback.print_exc()
    return "ok"
```

Gambar 2.5 Proses bot

Gambar 2.5 memperlihatkan contoh modul untuk mengatur proses *bot*. Proses yang dilakukan adalah mengambil data yang telah dikirim kemudian memecahkan data tersebut berdasarkan tipe data nya dengan menggunakan modul messaging_events. Setelah itu, data pesan akan di proses dalam modul processIncoming dan menghasilkan *output* yang disimpan dalam variabel response. *Variable* ini yang menentukan tipe pengembalian ke *bot*. Di dalam kasus ini, tipe pengembalian yang digunakan adalah *button* dan *text*. Modul yang dipakai adalah modul send_message dan send_button. Modul send_message digunakan untuk mengirim data *text* ke *bot*. Sedangkan modul send_button digunakan untuk mengirim data *button* ke *bot*.

```
def messaging_events(payload):

    data = json.loads(payload)
    messaging_events = data["entry"][0]["messaging"]

    for event in messaging_events:
        sender_id = event["sender"]["id"]

        # Not a message
        if "message" not in event:
            yield sender_id, None

        # Pure text message
        if "message" in event and "text" in event["message"] and "quick_reply" not in event["
            message"]:
            data = event["message"]["text"].encode('unicode_escape')
            yield sender_id, {'type':'text', 'data': data, 'message_id': event['message']['mid']}

        # Message with attachment (location, audio, photo, file, etc)
        elif "attachments" in event["message"]:

            # Location
            if "location" == event['message']['attachments'][0]["type"]:
                coordinates = event['message']['attachments'][0]['payload']['coordinates']
                latitude = coordinates['lat']
                longitude = coordinates['long']

                yield sender_id, {'type':'location','data':[latitude, longitude],'message_id':
                    event['message']['mid']}
```

Gambar 2.6 Modul messaging_events

```
            # Audio
            elif "audio" == event['message']['attachments'][0]["type"]:
                audio_url = event['message']['attachments'][0]['payload']['url']
                yield sender_id, {'type':'audio','data': audio_url, 'message_id': event['message']
                    ['mid']}

            else:
                yield sender_id, {'type':'text','data':"I don't understand this", 'message_id':
                    event['message']['mid']}

        # Quick reply message type
        elif "quick_reply" in event["message"]:
            data = event["message"]["quick_reply"]["payload"]
            yield sender_id, {'type':'quick_reply','data': data, 'message_id': event['message']['
                mid']}

        else:
            yield sender_id, {'type':'text','data':"I don't understand this", 'message_id': event[
                'message']['mid']}

# Allows running with simple `python <filename> <port>`
if __name__ == '__main__':
    if len(sys.argv) == 2: # Allow running on customized ports
        app.run(port=int(sys.argv[1]))
    else:
        app.run() # Default port 5000
```

Gambar 2.7 Modul messaging_events

Dalam modul messaging_events, pesan yang telah diterima oleh dipecah berdasarkan tipe datanya, seperti yang dijelaskan pada Gambar 2.6 dan Gambar 2.7. Disini data diubah ke dalam bentuk JSON yang lebih ringkas, untuk mempermudah proses selanjutnya.

```
def processIncoming(user_id, message):
    if message['type'] == 'text':
        message_text = message['data']

        if message['data'] == 'button!':
            message_text = 'button'

        return message_text

    elif message['type'] == 'location':
        response = "I've received location (%s,%s) (y)"%(message['data'][0],message['data'][1])
        return response

    elif message['type'] == 'audio':
        audio_url = message['data']
        return "I've received audio %s"%(audio_url)

    # Unrecognizable incoming, remove context and reset all data to start afresh
    else:
        return "*scratch my head*"
```

Gambar 2.8 Modul processIncoming

Gambar 2.8 menunjukan *source code* modul processIncoming yang berfungsi untuk menentukan apa yang akan dihasilkan dari data yang di-*input*-kan. Dalam kasus ini, data *text* yang di kirim *user* ke dalam *bot* akan di lemparkan kembali ke *user*, kecuali data 'button!'. Jika *user* meng-*input*-kan 'button!' maka bot akan mengembalikan data berupa *button*.

```
def send_message(token, user_id, text):
    """Send the message text to recipient with id recipient.
    """
    r = requests.post("https://graph.facebook.com/v2.6/me/messages",
                      params={"access_token": token},
                      data=json.dumps({
                          "recipient": {"id": user_id},
                          "message": {"text": text.decode('unicode_escape')}
                      }),
                      headers={'Content-type': 'application/json'})
    if r.status_code != requests.codes.ok:
        print r.text
```

Gambar 2.9 Modul send_message

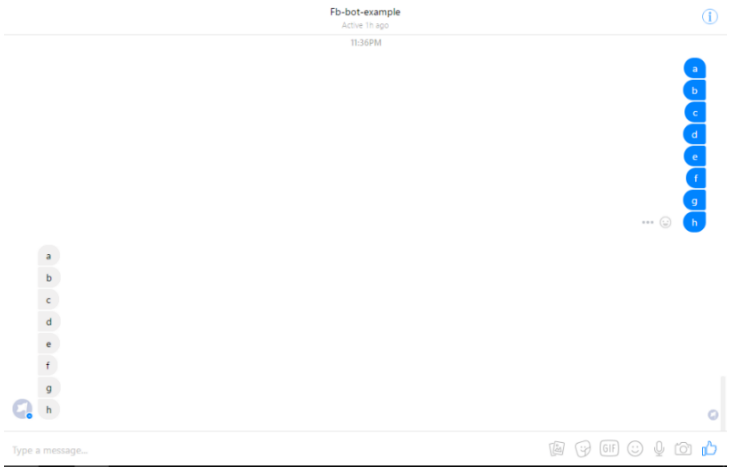
Pada Gambar 2.9 dijelaskan modul send_message yang digunakan untuk memanggil Facebook Messenger API yang berfungsi untuk mengirim data berupa *text*.

```
def send_button(token, user_id):
    r = requests.post("https://graph.facebook.com/v2.6/me/messages",
                      params={"access_token": token},
                      data=json.dumps({
                          "recipient": {"id": user_id},
                          "message": {
                                "attachment":{
                                  "type":"template",
                                  "payload":{
                                    "template_type":"button",
                                    "text":"What do you want to do next?",
                                    "buttons":[{
                                        "type":"web_url",
                                        "url":"https://petersapparel.parseapp.com",
                                        "title":"Show Website"
                                      },{
                                        "type":"postback",
                                        "title":"Start Chatting",
                                        "payload":"USER_DEFINED_PAYLOAD"
                                      }
                                    ]
                                  }
                                }
                            }
                      }),
                      headers={'Content-type': 'application/json'})
    if r.status_code != requests.codes.ok:
        print r.text
```

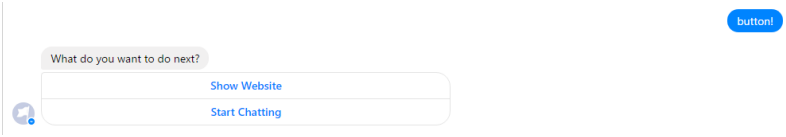
Gambar 2.10 Modul send_button

Pada Gambar 2.10 dijelaskan modul send_button yang digunakan untuk memanggil salah satu *resource* dari Facebook Messenger API yang berfungsi untuk mengirim data berupa *button*.

Hasil dari *chat* yang ditampilkan di Facebook Messenger dalam bentuk *text* biasa dapat dilihat pada Gambar 2.11, sedangkan untuk hasil dari *chat* berupa *button* dapat dilihat pada Gambar 2.12



Gambar 2.11 Pengembalian data *text*



Gambar 2.12 Pengembalian data *button*

2.4. Representational State Transfer (REST)

Salah satu bentuk interaksi antara *user* dan Internet adalah melalui *web aplication* dengan menggunakan *web service* yang menggunakan konsep Service Oriented Architecture (SOA). Salah satu *framework* yang digunakan dalam *web service* adalah Represantational State Transfer atau yang sering disebut REST. (Wagh, K. & Thool, R. , 2012). REST menggunakan teknologi yang lebih baru dibandingkan dengan pesaingnya, Simple Object Access Protocol (SOAP). Sebuah RESTful *web service* menggunakan Universal Resource Identifiers (URI) untuk mengakses *resource* yang ada dalam *web service*. *Resource* dapat dimanipulasi menggunakan 4 operasi yaitu: PUT, GET, POST, dan DELETE. PUT digunakan untuk membuat *resource* baru yang juga dapat dihapus menggunakan DELETE.

GET digunakan untuk mengambil *resource* dan POST digunakan untuk mengganti status *resource*. Dari keempat operasi tersebut hanya dua operasi yang paling sering digunakan, yaitu operasi GET dan POST. REST juga menggunakan konsep *self-descriptive message*, artinya setiap pesan atau *output* mengandung informasi bagaimana cara memproses pesan tersebut. Setiap interaksi dengan *resource* bersifat *stateless*, artinya setiap *request* mengandung status atau *state* yang dibutuhkan untuk menangani *request* tersebut baik di dalam URI, *query-string parameters, request headers* maupun *request body*. Selain itu setiap *response* juga mengandung status atau *state* dari *resource* baik di dalam *body, response codes*, maupun *response headers*. (Pautasso, C., Zimmermann, O., & Leymann, F., 2008).

Authentication dalam REST memerlukan 2 langkah. Langkah pertama adalah membuat dan mengatur *user credentials* pada *web service*. Langkah selanjutnya adalah memasukan *user credentials* setiap kali me-*request web service*. Hal ini dilakukan karena *web service* bersifat *stateless* sehingga *web service* tidak bisa mengingat *user* mana yang sudah *login*. Terdapat beberapa macam *authentication* dalam REST. Diantaranya adalah HTTP Basic, HTTP Digest, WSSE (Web Service Security Extension) Username Token dan OAuth. (Richardson, L., & Amundsen, M., 2013).

HTTP Basic akan mengirim *username* dan *password* yang telah di gabung dan di *encode* dengan Base64 *encoding*. *Authentication* dengan cara ini tidak disarankan karena *username* dan *password* yang di-*encode* dengan Base64 *encoding* dapat di kembalikan lagi menjadi *plain text*. Karena itu, siapapun yang dapat melihat koneksi Internet dari *user* akan mengetahui *username* dan *password user*. Masalah ini akan terpecahkan ketika *web service* menggunakan HTTPS dan bukan HTTP, karena jika menggunakan HTTPS, *request* dan *response* yang dikirim dan didapatkan *user* akan dienkripsi oleh SSL (Secure Sockets Layer). (Richardson, L., & Amundsen, M., 2013).

HTTP Digest akan mengirim suatu *string* yang biasa disebut digest pada *server*. *Client* tidak mengirim *password* secara langsung namun mengirim *string* yang telah di enkripsi supaya *server* tahu bahwa *client* mengetahui *password user*. HTTP Digest juga membutuhkan *client* untuk me-*request server* dua kali. *Request* yang pertama adalah untuk mendapatkan nonce. Nonce ini diperlukan *client* untuk

membuat digest. Digest dibuat dengan menghitung nilai MD5 dari nilai MD5 *user credentials* ditambahkan dengan nounce dan nilai MD5 dari metode yang digunakan dalam me-*request web service* serta *path* nya. Jika menggunakan HTTP Digest, *server* tidak akan mengetahui *password user*. Hal ini dikarenakan *server* telah menyimpan nilai MD5 dari *user credentials*, sehingga *server* dapat menghitung hasil akhir digest. (Richardson, L., & Ruby, S., 2007)