BAB 3
ANALISIS DAN PERANCANGAN SISTEM

3.1. Analisis Sistem

Analisis sistem adalah penguraian dari suatu sistem informasi yang utuh ke dalam bagian-bagian komponennya dengan maksud untuk mengidentifikasikan dan mengevaluasi permasalahan, kesempatan, hambatan yang terjadi dan kebutuhan yang diharapkan sehingga dapat diusulkan perbaikan merupakan tahap dalam analisis sebuah sistem.

3.1.1. Analisis Masalah

Berdasarkan masalah yang diuraikan di bab pendahuluan, maka analisis masalah dapat dijabarkan sebagai berikut:

1. Dibutuhkannya alternatif pembelajaran Bahasa Inggris yang mudah dan praktis. Disebabkan Bahasa Inggris adalah bahasa internasional yang sangat dibutuhkan dalam berbagai segi kehidupan. Kemampuan Bahasa Inggris yang baik juga akan menjadi modal kompetitif, baik dalam bidang pendidikan maupun pekerjaan. Solusi yang ditawarkan ialah sebuah aplikasi *chatbot* untuk melatih percakapan Bahasa Inggris.
2. Untuk melatih percakapan Bahasa Inggris diperlukan lawan bicara yang baik. Maksud baik disini ialah semakin sering berlatih percakapan dengan lawan bicara tersebut semakin meningkat kemampuan yang dimiliki. Misalnya setiap melakukan kesalahan maka si lawan bicara akan mengoreksi. Adapun solusi yang ditawarkan adalah adanya fitur koreksi pada aplikasi *chatbot* yang dibangun. Ketika terdapat kesalahan struktrur kalimat pada pesan *user*, maka *bot* akan memberikan koreksi.
3. Percakapan pada umumnya dilakukan dengan 2 cara, secara lisan atau tulisan, sehingga keduanya perlu dilatih. Dengan begitu pengguna akan terbiasa melakukan percakapan baik melalui teks(*Text messaging*) ataupun percakapan langsung dengan suara. Untuk itu aplikasi *chatbot* yang akan dibangun menyediakan 2 mode percakapan, satu diantaranya melalui *voice chat*, sedangkan satu yang lain melalui *chat(Text messaging)*.

4. Sebagai media latihan, diperlukan adanya koreksi dan evaluasi sehingga dapat terus meningkatkan kemampuannya dan tidak mengulangi kesalahan yang sama. Oleh sebab itu, aplikasi *chatbot* akan menyediakan fitur yang menampilkan log harian pengguna dalam melakukan percakapan dengan *bot*. Log harian tersebut berisi informasi tentang banyaknya *request*, koreksi, topik yang diambil, dan juga menggambarkan grafik perkembangan selama seminggu.

3.1.2. Solusi yang Dibangun

Berdasarkan analisis terhadap masalah yang ada, maka diperlukan sebuah media yang dapat digunakan untuk berlatih percakapan bahasa inggris dalam upaya meningkatkan kefasihan, pengetahuan grammar atau kosakata, dan kepercayaan diri dalam melakukan percakapan Bahasa Inggris. Media tersebut ialah berupa sebuah aplikasi *chatbot*. Teknologi *chatbot* dipilih karena dalam penerapannya dapat bertindak selayaknya manusia atau dalam konteks ini ialah sebagai lawan bicara.

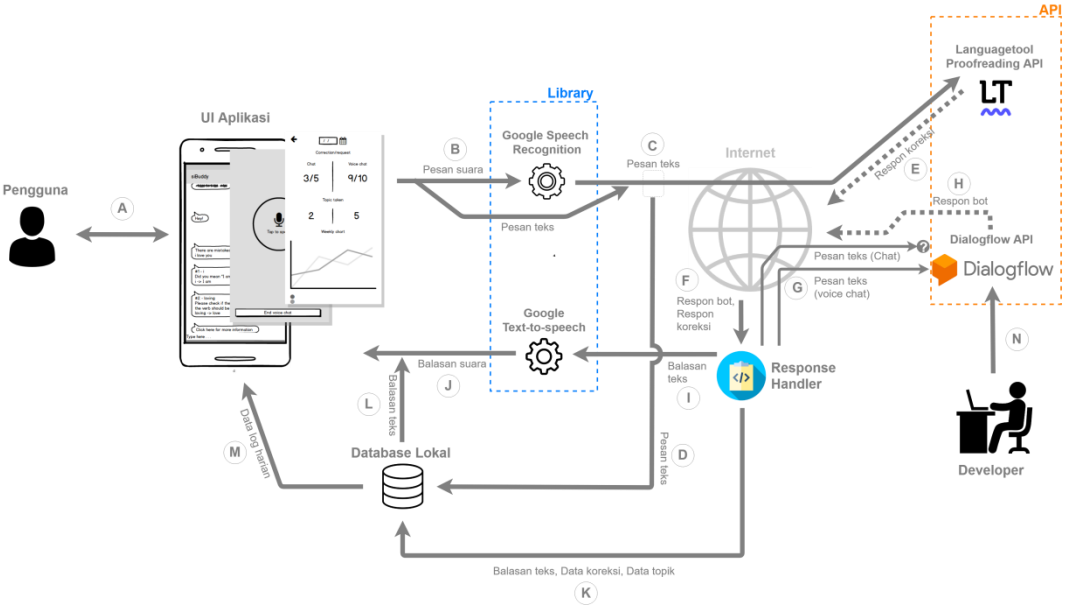
Adapun karakter *chatbot* yang dibangun adalah:

1. Pengguna dapat berinteraksi dengan *bot* mengenai beberapa topik bahasan sehari-hari dalam Bahasa Inggris seolah-oleh pengguna sedang melakukan percakapan dengan *native speaker*.
2. Aplikasi ini juga menyediakan dua mode percakapan, yakni mode *chat(texting)* dan mode *voice* atau suara**.** Dengan begitu, pengguna dapat melatih baik lisan maupun tulisannya.
3. Disertakannya fitur koreksi grammar dan log harian, sehingga pengguna akan terbantu dalam grammarnya sekaligus memudahkan pengguna untuk evaluasi mandiri.

3.1.3. Deskripsi Sistem

Sistem yang dibangun berupa aplikasi *chatbot*, dimana *user* dapat membuat percakapan dengan *bot* melalui antarmuka yang dirancang menyerupai aplikasi *chatting/text messaging* pada umumnya, sedangkan untuk *voice chatting*, antarmuka akan menyerupai aplikasi *voice assistance*. Sistem akan merekam aktifitas harian user untuk nantinya dibuatkan log harian beserta grafik log per

minggu sebagai sarana evaluasi mandiri bagi *user*. Adapun cara kerja sistem secara umum ditunjukan oleh gambar 3.1.



Gambar 3.1 Deskripsi Sistem

Penjelasan untuk Gambar 3.1 ialah sebagai berikut:

A. Interaksi *User – UI* Aplikasi

Dsini terjadi interaksi antara *user* dan antarmuka aplikasi, bentuk interaksinya antara lain *user* melakukan *chat* dengan *bot*, *user* melakukan *voice chat*, dan *user* melihat log harian. Ketika user melakukan *chat*, *user* disuguhkan daftar *chat* terakhir dan *form* kirim pesan. Sedangkan ketika *user* melakukan *voice chat*, *user* disuguhkan dengan sebuah tombol untuk memulai berbicara. Adapun ketika *user* melihat log harian, *user* disuguhkan informasi berupa jumlah *request*, koreksi, dan topik yang diambil pada hari tertentu, dan grafik log harian per minggu.

B. Konversi Pesan Suara ke Teks

Ketika *user* melakukan *voice chat*, maka pesan *user* yang berupa rekaman audio dikonversi menjadi teks (*String*) dengan bantuan *library Google Speech Recognition*. Fungsi *record* audio sudah termasuk di dalam *Google Speech Recognition*.

C. *Request* Koreksi ke *Languagetool(Proofreading API)*

Di mode *voice chat*, ketika pesan user telah dikonversi menjadi teks, maka pesan tersebut akan diteruskan ke *proofreading API(Languatool)* untuk meminta koreksi terhadap teks tersebut. Adapun di mode *chat*, pesan *user* sudah berupa teks sehingga pesan tesebut dapat langsung diteruskan ke *proofreading API*. Proses ini membutuhkan koneksi internet.

D. Tulis Pesan ke *Database*

Pesan *user* baik yang diperoleh pada mode *chat* ataupun pada mode *voice chat*(yang telah dikonversi) disimpan ke *database*.

E. *Languagetool* Memberikan Respon

Segera setelah *request* koreksi, *proofreading API* akan memberikan respon dalam format JSON, respon berisi informasi antara lain, pesan kesalahan, letak kesalahan, dan saran penggantian kata.

F. *Parsing* Respon dari API

Disini setiap respon dari API*(Languagetool* atau *Dialogflow*) akan di uraikan menjadi data dalam format yang sederhana. Terdapat beberapa objek yang bekerja untuk proses ini, akan dijelaskan pada bagian arsitektur program. Adapun output dari proses ini antara lain pesan balasan(*String*), data koreksi, data topik.

G. *Request* Balasan terhadap Pesan *User* ke *Dialogflow*

Di mode *voice chat*, setelah respon koreksi diperoleh, maka selanjutnya pesan *user* diteruskan ke *chatbot API* (*Dialogflow*) untuk meminta balasan terhadap pesan tersebut. Sedangkan di mode *chat*, proses ini dilaksanakan hanya jika tidak terdapat koreksi pada proses sebelumnya(*Request* koreksi ke *languagetool*).

H. *Dialogflow* Memberikan Respon

Segera setelah request balasan, *Chatbot API* akan memberikan respon yang disimpan dalam variabel bertipe data AIResult(kelas dalam paket *Dialogflow API*).

I. Konversi Balasan Teks ke Suara

Di mode *voice chat*, segera setelah respon dari *chatbot API* di uraikan ke dalam format sederhana, maka pesan balasan(Teks) hasil penguraian langsung di konversi menjadi suara dengan memanfaatkan *library Google Text-to-speech*. Setelah itu, barulah pesan balasan disimpan ke *database*. Sedangkan di mode *chat*, pesan balasan hasil penguraian akan disimpan terlebih dahulu ke *database*.

J. Memutar Balasan Suara

Setelah proses konversi teks – suara, maka selanjutnya balasan(suara) diputar. Fungsi ini sudah termasuk di dalam *Google Speech-to-text*.

K. Tulis Balasan Teks, Data Koreksi, dan Topik ke *Database*

Setelah respon selesai diurai kedalam format sederhana, maka data-data tersebut akan disimpan ke *database*.

L. Menampilkan Balasan Teks

Proses ini dilakukan hanya di mode *chat*. Pesan balasan akan dibaca dari *database*, kemudian pesan tersebut ditampilkan dalam daftar chat terakhir.

M. Baca Log Harian dari *Database*

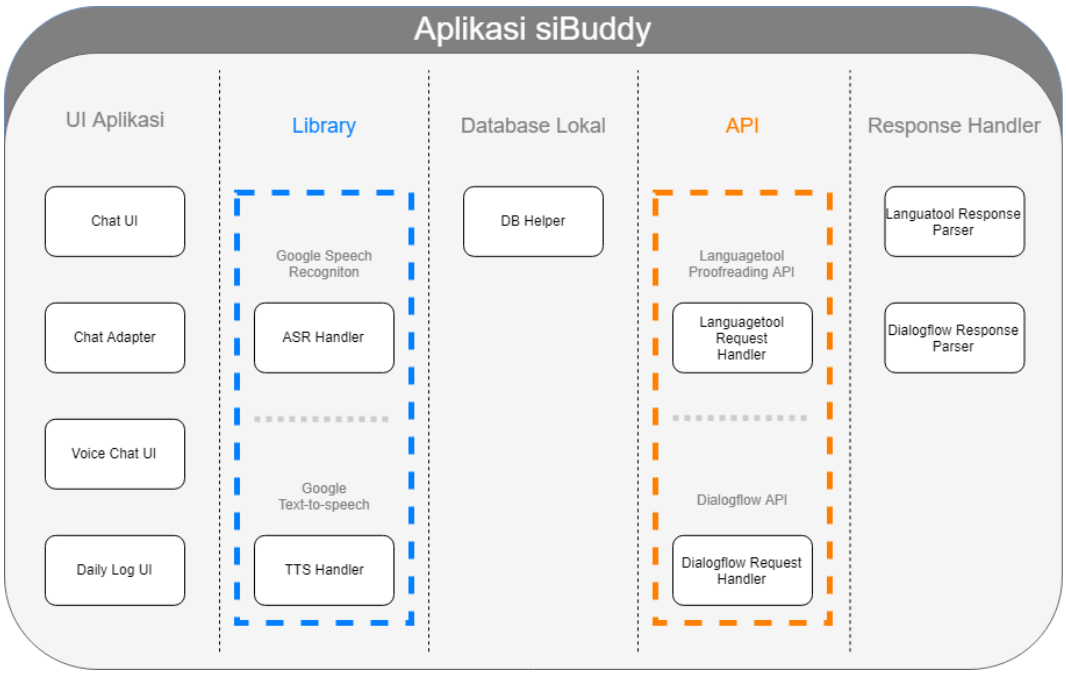
Proses ini akan dilakukan ketika *user* melihat log harian, data log harian yang berupa banyak *request*, koreksi, dan topik yang diambil akan dibaca dari *database* untuk ditampilkan ke antarmuka log harian. *User* dapat mengatur log harian untuk tanggal berapa yang ingin ditampilkan.

N. *Update Knowledge Base Dialogflow* oleh Developer

Disini developer akan secara berkala melakukan *update* terhadap *knowledbase Dialogflow API* melalui *dialogflow developer console*.

3.1.4. Arsitektur Sistem yang Dibangun

Sistem terdiri dari 5 bagian utama, yakni UI Aplikasi, *Library(Google Speech Recognition & Google Text-to-speech*), *Database* Lokal(*Sqlite*), API(*Languagetool & Dialogflow*), dan *Response Handler*. Untuk lebih jelasnya lihat gambar 3.2.



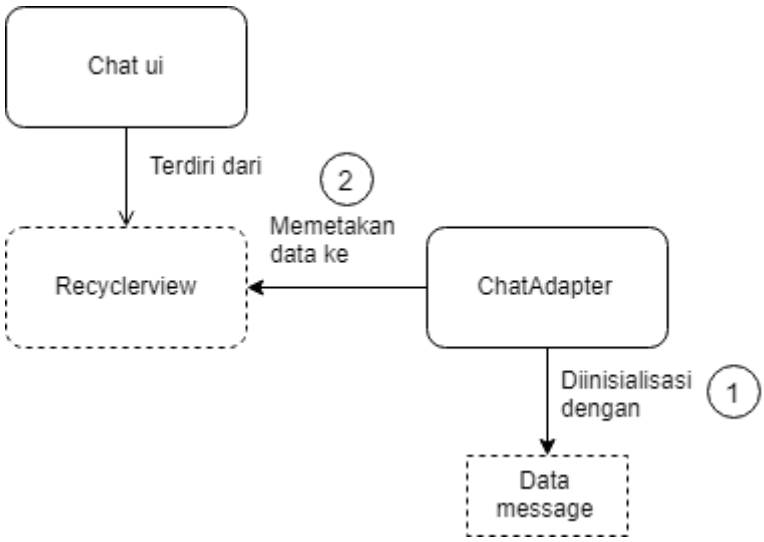
Gambar 3.2 Arsitektur Sistem

Adapun penjelasan tiap-tiap bagiannya adalah sebagai berikut:

1. UI Aplikasi

   Antarmuka antara pengguna dengan aplikasi.

   a. *Chat UI*, ialah antarmuka untuk melakukan percakapan dengan bot melalui teks. Di dalamnya terdapat daftar *chat* terakhir(*recyclerview*) dan *form* kirim pesan(*edittext*). Untuk menampilkan pesan pada daftar *chat* terakhir maka diperlukan sebuah objek *Chat Adapter* yang mana merupakan turunan dari kelas *Adapter*. UI ini diimplementasikan dengan kelas ChatActivity turunan kelas Activity. Dibawah ini memperlihatkan hubungan antara *Chat UI* dan *Chat Adapter* dalam menampilkan daftar *chat* terakhir, ditunjukan oleh gambar 3.3.



Gambar 3.3 Chat UI dan Chat Adapter

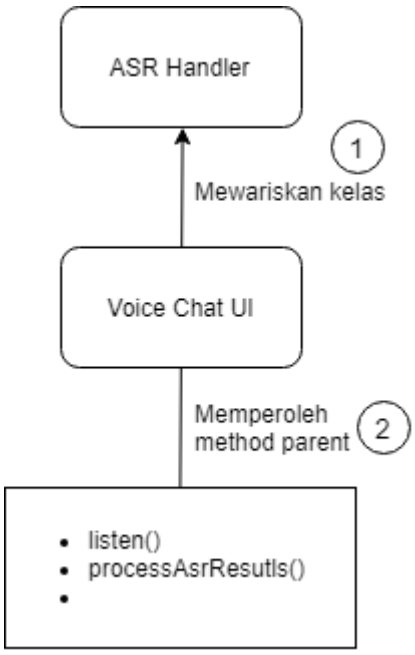
b. *Voice Chat UI*, ialah antarmuka untuk melakukan percakapan dengan *bot* melalui suara, didalamya disediakan tombol(*button*) untuk memulai *record* audio. Adapun ketika respon balasan terhadap pesan telah diperoleh maka balasan berupa suara/audio tersebut langsung diputar. Segera setelah balasan suara diputar, maka aplikasi secara otomatis langsung melakukan *record* untuk *request* selanjutnya. UI ini diimplementasikan dengan kelas VoiceChatFragment turunan kelas Fragment.

c. *Daily Log UI*, ialah antarmuka yang menyajikan log harian kepada pengguna berikut grafik perkembangannya selama seminggu kebelakang. Disediakan pula kolom tanggal(*datepicker*) untuk memilih data harian pada tanggal berapa yang ingin ditampilkan. UI ini diimplementasikan dengan kelas DailyLogActivity turunan kelas Activity.

2. *Library*

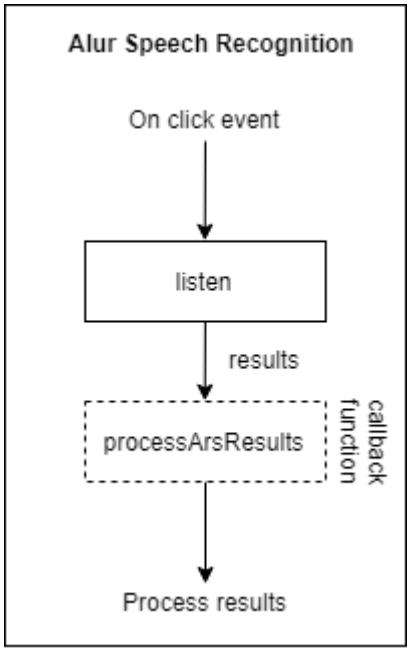
Di mode voice chat, aplikasi ini memanfaatkan library untuk mengkonversi suara ke teks begitupun sebaliknya, teks ke suara. *Library* yang digunakan ialah *Google Speech Recognition* dan *Google Text-to-speech.*

a. *Google Speech Recognition*, untuk menggunakan *library* ini, penulis membuat sebuah objek *ASR Handler*, objek ini akan mengimplementasikan fungsi-fungsi yang terdapat pada *library google speech recognition*, termasuk juga diantaranya beberapa fungsi *callback*. Adapun fungsi utama yang dimiliki objek ini ialah fungsi `Listen()`, sudah termasuk didalamnya fungsional untuk *record* audio. Fungsi ini dipanggil ketika user menekan tombol *speak* pada *Voice Chat UI*. Untuk lebih jelasnya lihat gambar 3.4.



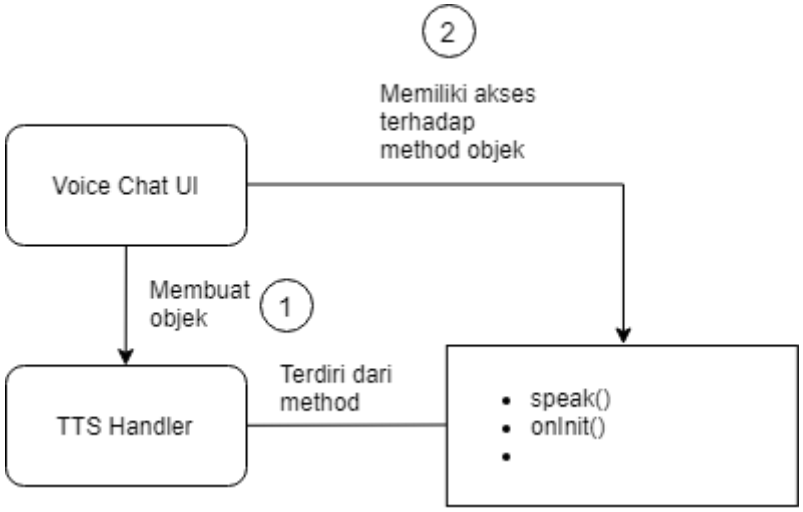
Gambar 3.4 *ASR Handler*

Adapun alur proses *speech recognition* ditunjukan oleh gambar 3.5.



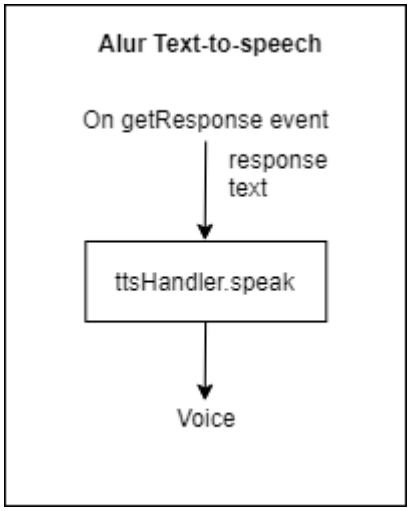
Gambar 3.5 Alur *Speech Recognition*

b. *Google Text-to-speech*, sama seperti *Google Speech recognition*, untuk menggunakan library ini, dibuatkan pula sebuah objek untuk mengimplementasikan fungsi-fungsinya. Objek tersebut ialah *TTS Handler*. Fungsi utama yang dimiliki objek ini ialah fungsi `Speak()`, yang mana sudah termasuk didalamnya fungsional untuk memutar audio segera setelah teks dikonversi menjadi suara. Fungsi `speak()` dipanggil ketika respon terhadap pesan *user* sudah diterima. Untuk lebih jelasnya lihat gambar 3.6.



Gambar 3.6 *TTS Handler*

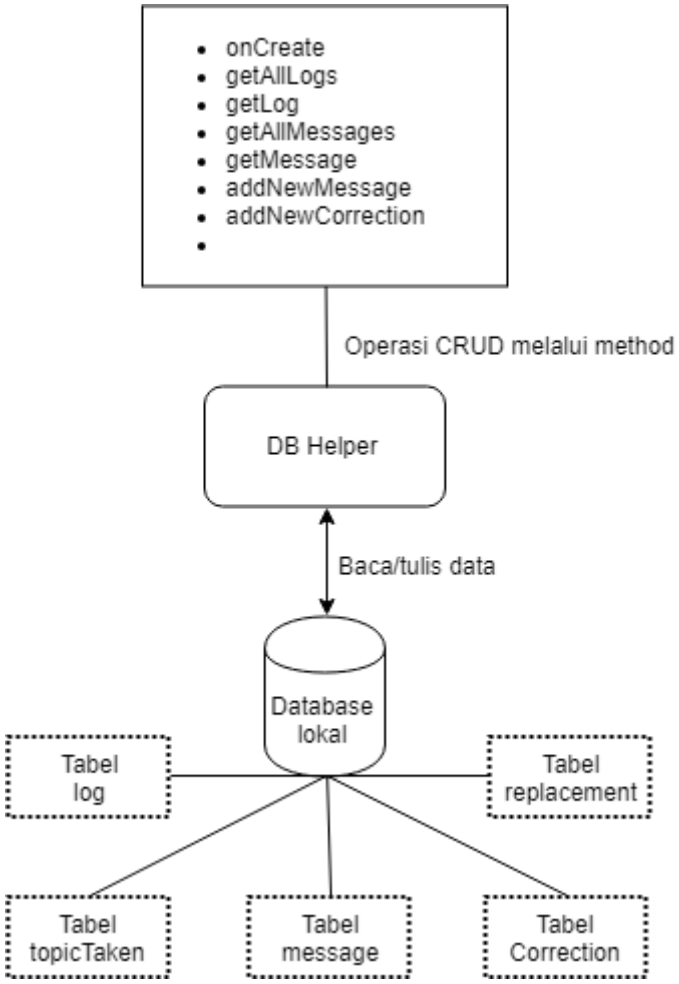
Adapun alur proses *text-to-speech* ditunjukan oleh gambar 3.7.



Gambar 3.7 Alur *Text-to-speech*

3. *Database* Lokal

   Aplikasi ini menggunakan *Sqlite* sebagai *database* lokal untuk menyimpan atau membaca data pesan/balasan, koreksi, *replacement* dan *topic taken*. Untuk menjalankan fungsi, terdapat objek *DB Helper* yang merupakan turunan dari kelas SQLiteOpenHelper, yang mana objek ini memiliki fungsi untuk *create database* dan *drop database*, termasuk juga fungsi untuk *insert*, *update*, *delete*, dan *query* untuk setiap tabel yang disertakan disini. Untuk lebih jelasnya lihat gambar 3.8.



Gambar 3.8 DB Helper

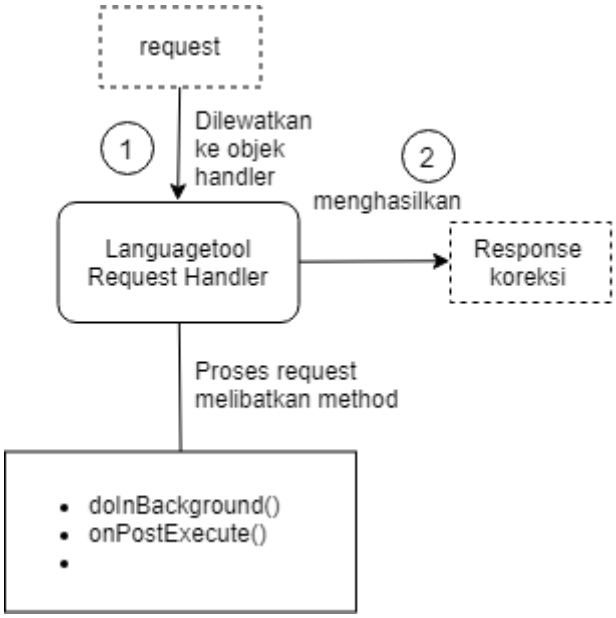
4. API

   Aplikasi ini memanfaatkan API untuk menjalankan fungsinya. API *languagetool* akan digunakan untuk menangani *request* koreksi terhadap pesan *user*, sedangkan API *Dialogflow* akan digunakan untuk menangani *request* balasan terhadap pesan *user*, dengan kata lain membuat *flow* percakapan. Untuk menggunakan API *languagetool* ataupun *Dialogflow*, masing-masing dibuatkan sebuah objek untuk menangani *request* ke kedua API tersebut.

   a) Languagetool Proofreading API

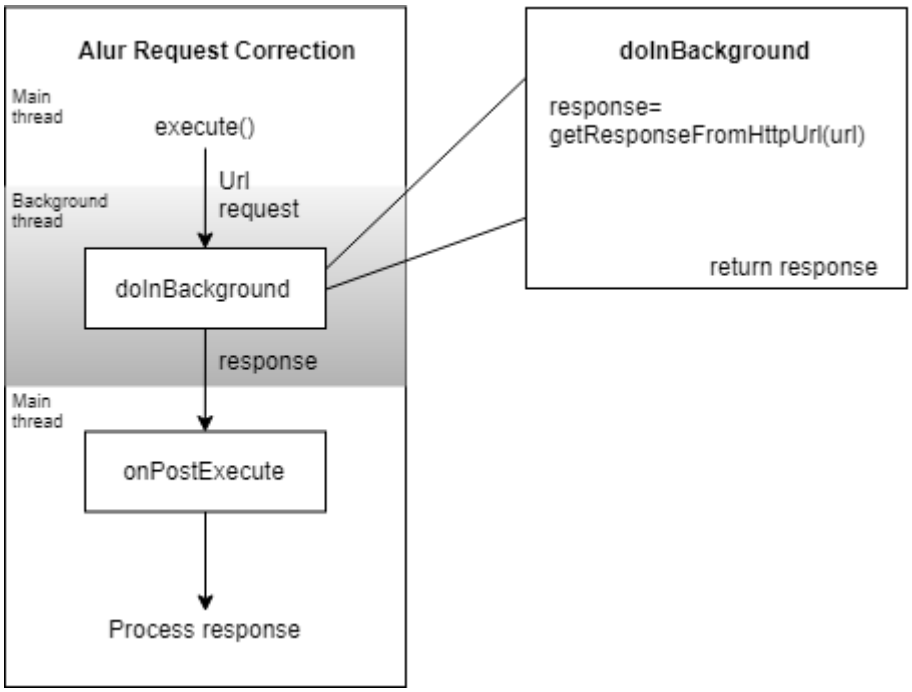
   Request koreksi ke API *languagetool* ditangani oleh objek *Languagetool Request Handler*. Objek ini memiliki fungsi untuk membuat koneksi ke API dan mendapatkan respon koreksi dalam

format JSON, koneksi ini tidak memerlukan sebuah *API Key*. Proses ini berjalan di *background*. Adapun parameter yang dikirim ialah teks pesan itu sendiri dan bahasa yang digunakan. Untuk lebih jelasnya lihat gambar 3.9



Gambar 3.9 *Languagetool Request Handler*

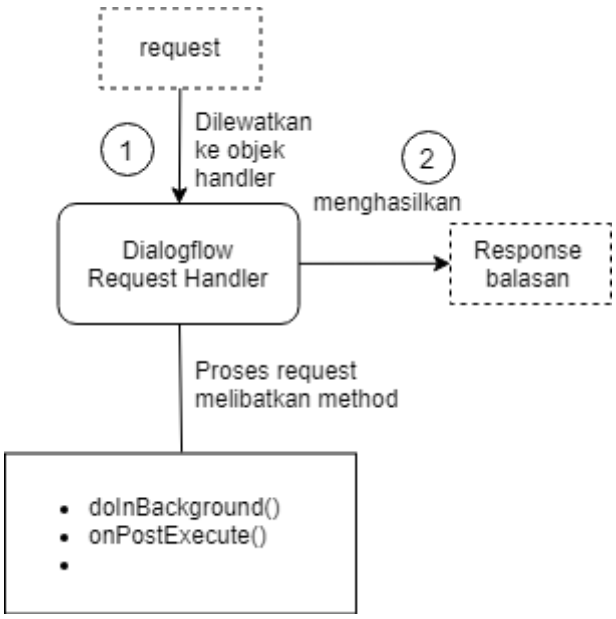
Adapun alur proses *request correction* lihat gambar 3.10



Gambar 3.10 Alur *Request Correction*

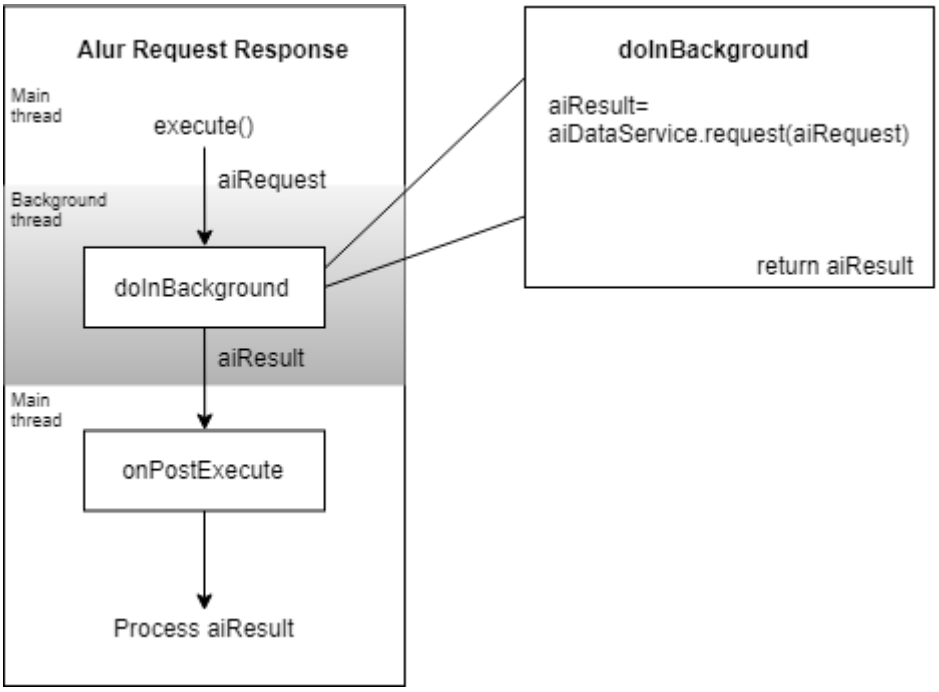
b) Dialogflow API

Request balasan ke API *dialogflow* ditangani oleh *Dialogflow Request Handler*. *Dialogflow* API sendiri dalam paketnya telah menyediakan kelas-kelas untuk penanganan *request*. Proses *request* dimulai dari inisialisasi objek bertipe AIDataService. Pada proses inisialisasi, pengembang diminta memasukan *API key dialogflow* yang diperoleh melalui *dialogflow developer console*. Selanjutnya objek bertipe AIDataService memanggil fungsi `request()` dengan melewatkan parameter berupa teks pesan. Proses ini berjalan di *background*. Untuk lebih jelasnya lihat gambar 3.11



Gambar 3.11 *Dialogflow Request Handler*

Adapun untuk alur proses *request reply* ditunjukan oleh gambar 3.12.



Gambar 3.12 Alur *Request Reply*

5. *Respon Handler*

   Setelah *request* ke API baik *languagetool* ataupun *dialogflow*, maka setelahnya akan menerima respon. Respon tersebut akan ditangani oleh objek yang berfungsi untuk menguraikan respon ke dalam format yang lebih sederhana. Objek-objek itu ialah:

   a) Languagetool Response Parser

   Objek ini akan menangani respon dari API *languagetool* yang berupa informasi koreksi dalam format JSON. Informasi tersebut kemudian diurai ke dalam format sederhana, yakni *String* dan *integer*. Berikut ini adalah contoh respon dari *languagetool*:

```
},
  "matches":[
     {
        "message":"Did you mean \"I am\"?",
        "shortMessage":"",
        "replacements":[
           {
              "value":"I am"
           }
        ],
        "offset":0,
        "length":1,
        "context":{
           "text":"i loving you",
           "offset":0,
           "length":1
        },
        "sentence":"i loving you",
        "rule":{
           "id":"I_AM",
           "subId":"1",
           "description":"missing 'am' in 'I am'",
           "issueType":"grammar",
           "category":{
              "id":"GRAMMAR",
              "name":"Grammar"
           }
        }
     },
     {
        "message":"Please check if the base form of the verb
should be used here.",
        "shortMessage":"",
        "replacements":[
           {
              "value":"love"
```


Adapun informasi yang diuraikan antara lain:

1) "*message*" atau pesan kesalahan
2) "*replacements"* atau saran penggantian kata
3) "*offset*" dan "length" menunjukan letak kesalahan dalam kalimat
4) "*id*" dan "*subId*" untuk merujuk ke alamat yang disediakan *languagetool* untuk memberikan informasi lebih lanjut mengenai aturan *grammar* terkait. Contoh url nya sebagai berikut:

   https://community.languagetool.org/rule/show/[ID]?subId=[SUBID]&lang=en-US

b) *Dialogflow Response Parser*

Berfungsi untuk menguraikan respon dari API *dialogflow*. Setelah *request* diterima oleh *dialogflow* maka *dialogflow* akan mengirimkan respon, berbeda dengan *languagetool*, *dialogflow* dalam paketnya telah menyertakan sebuah kelas AIResponse, respon dari *dialogflow* akan disimpan dalam variabel bertipe kelas ini. Untuk mendapatkan informasi dari respon tersebut, developer hanya perlu memanggil fungsi untuk mendapatkan nilai tertentu. Misalnya untuk mendapatkan respon balasan teks dapat menggunakan fungsi `getResult().getFulfillment().getSpeech()` atau untuk mendapatkan *action* dapat menggunakan fungsi `getResult().getAction()`.

3.1.5. Analisis Teknologi yang Digunakan

Dalam pembangunan aplikasi siBuddy, penulis memanfaatkan beberapa *library* dan API, yaitu *Google Speech Recognition, Google Text-to-speech, Languagetool API,* dan *Dialogflow API.*

1. Google Speech Recognition

Secara default, library ini telah termasuk di dalam paket android SDK. Untuk menggunakannya terlebih dahulu penulis membuatkan kelas ASRHandler mewariskan kelas Activity dan mengimplentasikan *Interface* RecognizerIntent. RecognizerIntent mengharuskan ASRHandler mengimplementasikan beberapa fungsi *callback*, salah satunya ialah onResults() yang akan digunakan untuk mendapatkan hasil konversi.

```
1.  public abstract class ASRHandler extends Activity implements Recognition
    Listener{
2.
3.      @Override
4.      public void onResults(Bundle results) {
5.
6.      }
7.
8.      ...
9.  }
```

Langkah selanjutnya ialah buat fungsi untuk inisialisasi *instance* SpeechRecognizer. Di kelas ASRHandler buat fungsi createRecognizer() seperti terlihat pada potongan kode dibawah.

```
1.  private static SpeechRecognizer myASR;
2.  Context ctx;
3.
4.  public void createRecognizer(Context ctx) {
5.      this.ctx = ctx;
6.      PackageManager packManager = ctx.getPackageManager();
7.
8.      // find out whether speech recognition is supported
9.      List<ResolveInfo> intActivities = packManager.queryIntentActivities(

10.             new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);
11.     if (intActivities.size() != 0) {
12.         myASR = SpeechRecognizer.createSpeechRecognizer(ctx);
13.         myASR.setRecognitionListener(this);
14.     }
15.     else
16.         myASR = null;
17. }
```

Setelah itu, buat fungsi listen() untuk nantinya dipanggil ketika user menekan tombol speak. Di dalam fungsi inilah *instance* SpeechRecognizer memanggil fungsi untuk memulai *record* audio dan konversi suara ke teks.

```
1.  public void listen(String languageModel, int maxResults) throws Exceptio
    n{
2.
3.      if((languageModel.equals(RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
    || languageModel.equals(RecognizerIntent.LANGUAGE_MODEL_WEB_SEARCH)) &&
    (maxResults>=0)) {
4.          Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPE
    ECH);
5.
6.          // Specify the calling package to identify the application
7.          intent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE, ctx.getP
    ackageName());
8.          //Caution: be careful not to use: getClass().getPackage().getNam
    e());
9.
10.         // Specify language model
11.         intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, languageM
    odel);
12.
13.         // Specify how many results to receive. Results listed in order
    of confidence
14.         intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, maxResults);

15.         intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,"en-US");
16.
```

```
17.         // Start recognition
18.         myASR.startListening(intent);
19.     }
20.     else {
21.         Log.e(LIB_LOGTAG, "Invalid params to listen method");
22.         throw new Exception("Invalid params to listen method"); //If the
    input parameters are not valid, it throws an exception
23.     }
24.
25. }
```

Seperti yang sudah dijelaskan sebelumnya, hasil dari proses *speech recognition* diperoleh melalui fungsi *callback* onResult(), yakni disimpan dalam variabel results. Jika results tidak null maka akan memanggil fungsi processAsrResult() yang ada di VoiceChatActivity dan melewatkan variabel results sebagai parameter. Lihat potongan kode dibawah.

```
1.  @Override
2.  public void onResults(Bundle results) {
3.      Log.d(LIB_LOGTAG, "ASR results provided");
4.
5.      if(results!=null){
6.
7.          if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDW
    ICH) {  //Checks the API level because the confidence scores are support
    ed only from API level 14:
8.              //http://developer.android.com/reference/android/speech/Spee
    chRecognizer.html#CONFIDENCE_SCORES
9.              //Processes the recognition results and their confidences
10.             processAsrResults (results.getStringArrayList(SpeechRecogniz
    er.RESULTS_RECOGNITION), results.getFloatArray(SpeechRecognizer.CONFIDEN
    CE_SCORES));
11.             //Attention: It is not RecognizerIntent.EXTRA_RESULTS, that
    is for intents (see the ASRWithIntent app)
12.         }
13.         else {
14.             processAsrResults (results.getStringArrayList(SpeechRecogniz
    er.RESULTS_RECOGNITION), null);
15.         }
16.     }
17.     else
18.         processAsrError(SpeechRecognizer.ERROR_NO_MATCH);
19. }
```

Pembuatan kelas ASRHandler sudah selesai, selanjutnya adalah implementasi kelas ASRHandler di UI Voice Chat atau dalam kasus ini kelas VoiceChatActivity. Langkah pertama ialah mewariskan kelas ASRHandler dan melakukan *override* terhadap beberapa fungsi kelas parent, salah satunya

ialah processAsrResult() yang telah disinggung sebelumnya. Lihat potongan kode dibawah.

```
1.  public class VoiceChatActivity extends ASRHandler {
2.
3.      @Override
4.      public void processAsrResults(ArrayList<String> nBestList, float[] n
    BestConfidences) {
5.
6.      }
7.
8.      ...
9.  }
```

Langkah selanjutnya adalah menanamkan fungsi listen() di button speak dan panggil fungsi createRecognizer() yang telah dibuat sebelumnya di kelas ASRHandler untuk inisialisasi *instance* SpeechRecognizer.

```
1.  speak.setOnClickListener(new View.OnClickListener() {
2.      @Override
3.      public void onClick(View v) {
4.          //Speech recognition does not currently work on simulated device
    s,
5.          //it the user is attempting to run the app in a simulated device

6.          //they will get a Toast
7.          if("generic".equals(Build.BRAND.toLowerCase())){
8.              Toast toast = Toast.makeText(getApplicationContext(),"ASR is
     not supported on virtual devices", Toast.LENGTH_SHORT);
9.              toast.show();
10.             Log.d(LOGTAG, "ASR attempt on virtual device");
11.         } else{
12.             try {
13.                 //setRecognitionParams(); //Read ASR parameters
14.                 listen(languageModel, numberRecoResults); //Start listen
    ing
15.             } catch (Exception e) {
16.                 Toast toast = Toast.makeText(getApplicationContext(),"AS
    R could not be started: invalid params", Toast.LENGTH_SHORT);
17.                 toast.show();
18.                 Log.e(LOGTAG, "ASR could not be started: invalid params"
    );
19.             }
20.         }
21.     }
22. });
23.
24.
25. createRecognizer(getApplicationContext());
```

Langkah terakhir ialah memproses hasil koreksi melalui fungsi processAsrResult(). Hasil koreksi yang dipakai adalah hasil konversi dengan score terbesar, yakni terdapat pada indeks pertama atau nBestView.get(0).

```
@Override
public void processAsrResults(ArrayList<String> nBestList, float[] nBestConfidences) {
    changeButtonAppearanceToDefault(); //Button has its default appearance (so that the user knows the app is not listening anymore)

    //Creates a collection of strings, each one with a recognition result and its confidence, e.g. "Phrase matched (conf: 0.5)"
    ArrayList<String> nBestView = new ArrayList<String>();

    if(nBestList!=null){
        for(int i=0; i<nBestList.size(); i++){
            if(nBestConfidences!=null){
                if(nBestConfidences[i]>=0)
                    nBestView.add(nBestList.get(i) /*+ " (conf: " + String.format("%.2f", nBestConfidences[i]) + ")"*/);
                else
                    nBestView.add(nBestList.get(i) /*+ " (no confidence value available)"*/);
            }
        }
    }

    ...
}
```

2. Google Text-to-speech

Sama seperti *Google Speech Recognition*, Library ini juga sudah termasuk di dalam paket android SDK. Adapun untuk penggunaannya terlebih dahulu dibuatkan kelas yang TTSHandler dan mengimplementasikan interface onInitListener. Interface tersebut mengharuskan kelas TTSHandler mengimplementasikan fungsi callback onInit. Fungsi ini akan berguna ketika ingin mengetahui apakah teks telah selesai dibaca ataukah belum.

```
public class TTSHandler implements OnInitListener{

    ...

    @Override
    public void onInit(int status) {

    }

}
```

Langkah selanjutnya ialah inisialisasi *instance* TextToSpeech. Fungsional ini kita tuliskan dalam fungsi getInstance(). Lihat potongan kode dibawah.

```
1.  private TextToSpeech myTTS;
2.  private static TTS singleton;
3.
4.  private OnTtsDone listener;
5.  Activity activity;
6.
7.
8.  private TTS(Context ctx) {
9.      activity= (Activity)ctx;
10.     listener=(OnTtsDone) ctx;
11.     myTTS = new TextToSpeech(ctx,(OnInitListener) this);
12. }
13.
14. public static TTS getInstance(Context ctx){
15.     if(singleton==null){
16.         singleton = new TTS(ctx);
17.     }
18.
19.     return singleton;
20. }
```

Setelah itu, buat fungsi speak() yang akan dipanggil segera setelah respon dari API *Dialogflow* diterima.

```
1.  public void setLocale(){
2.      myTTS.setLanguage(Locale.getDefault());
3.  }
4.
5.  public void speak(String text, String languageCode) throws Exception{
6.      setLocale(languageCode);
7.      HashMap<String, String> map = new HashMap<String, String>();
8.      map.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID,"messageID");
9.      myTTS.speak(text, TextToSpeech.QUEUE_ADD, map);
10. }
```

Kelas TTSHandler telah selesai dibuat, selanjutnya implementasikan pada VoiceChatActivity. Langkah pertama ialah inisialisasi instance TextToSpeech dengan memanggil fungsi getInstance() dan simpan ke variabel myTts.

```
1.  myTts = TTS.getInstance(this);
```

Langkah selanjutnya cukup panggil fungsi speak() ketika respon telah diterima.

```
try {
    myTts.speak(aiResponse.getResult().getFulfillment().getSpeech(),"en-US");
} catch (Exception e) {
    e.printStackTrace();
}
```

3. Languagetool API

Untuk menggunakan API languagetool, maka langkah pertama ialah membuat koneksi ke languagetool. Adapun basis url adalah https://languagetool.org/api/v2/check dan variabel yang dikirim adalah *text* dan *language*. Untuk pembuatan url lihat potongan kode dibawah.

```
final static String LANGUAGETOOL_BASE_URL =
            "https://languagetool.org/api/v2/check";

final static String PARAM_TEXT = "text";
final static String PARAM_LANGUAGE = "language";

public static URL buildUrl(String text) {
    // COMPLETED (1) Fill in this method to build the proper Github query URL
    Uri builtUri = Uri.parse(LANGUAGETOOL_BASE_URL).buildUpon()
            .appendQueryParameter(PARAM_TEXT, text)
            .appendQueryParameter(PARAM_LANGUAGE, language)
            .build();

    URL url = null;
    try {
        url = new URL(builtUri.toString());
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }

    return url;
}
```

Adapun untuk proses request dilakukan oleh objek LanguagetoolRequestHandler. Proses ini akan membutuhkan waktu sehingga proses ini akan dijalankan di *background*. Oleh karena itu, kelas ini akan mewariskan kelas Asynctask.

```
1.  public class LanguagetoolRequestHandler extends AsyncTask<URL, Void, Cor
    rection[]> {
2.
3.      @Override
4.      protected Correction[] doInBackground(URL... params) {
5.
6.      }
7.
8.      @Override
9.      protected void onPostExecute(Correction[] githubSearchResults) {
10.
11.     }
12. }
```

Proses *request* akan dijalankan di dalam fungsi doInBackground() dan respon disimpan dalam variabel languagetoolJsonResponse.

```
1.  public class LanguagetoolRequestHandler extends AsyncTask<URL, Void, Cor
    rection[]> {
2.
3.      @Override
4.      protected Correction[] doInBackground(URL... params) {
5.          URL searchUrl = params[0];
6.          String languagetoolJsonResponse = null;
7.          Correction[] corrections = null;
8.
9.          try {
10.             languagetoolJsonResponse = getResponseFromHttpUrl(searchUrl)
    ;
11.
12.         } catch (IOException e) {
13.             e.printStackTrace();
14.         } catch (JSONException e) {
15.             e.printStackTrace();
16.         }
17.     }
18.
19.     @Override
20.     protected void onPostExecute(Correction[] corrections) {
21.
22.     }
23.
24.     public String getResponseFromHttpUrl(URL url) throws IOException {
25.         HttpURLConnection urlConnection = (HttpURLConnection) url.openCo
    nnection();
26.         try {
27.             InputStream in = urlConnection.getInputStream();
28.
29.             Scanner scanner = new Scanner(in);
30.             scanner.useDelimiter("\\A");
31.
32.             boolean hasInput = scanner.hasNext();
33.             if (hasInput) {
34.                 return scanner.next();
35.             } else {
36.                 return null;
```

```
            }
        } finally {
            urlConnection.disconnect();
        }
    }
}
```

Selanjutnya buat kelas LanguagetoolResponseParser yang memiliki fungsi *static* untuk mengurai respon *languagetool* ke dalam bentuk yang sederhana.

```
public class LanguagetoolResponseParser {

    public static Correction[] getSimpleDataFromJson(Context context, St
ring languagetoolResponseJsonStr, int messageId)
            throws JSONException {

        final String LT_MATCHES = "matches";

        final String LT_MESSAGE = "message";
        final String LT_REPLACEMENTS = "replacements";

        final String LT_REPLACEMENTS_VAL = "value";

        final String LT_OFFSET = "offset";
        final String LT_LENGTH = "length";
        final String LT_SENTENCE = "sentence";
        final String LT_RULE = "rule";

        final String LT_RULE_ID = "id";
        final String LT_RULE_SUB_ID = "subId";
        final String LT_RULE_DESC = "description";


        Correction[] corrections = null;

        JSONObject languagetoolResponseJson = new JSONObject(languagetoo
lResponseJsonStr);

        JSONArray correctionArray = languagetoolResponseJson.getJSONArra
y(LT_MATCHES);

        corrections = new Correction[correctionArray.length()];

        for (int i = 0; i < correctionArray.length(); i++) {
            /* These are the values that will be collected */
            String message;
            String[] replacements = null;
            int offset;
            int length;
            String sentence;
            String ruleId;
            int ruleSubId;
            String ruleDesc;

```

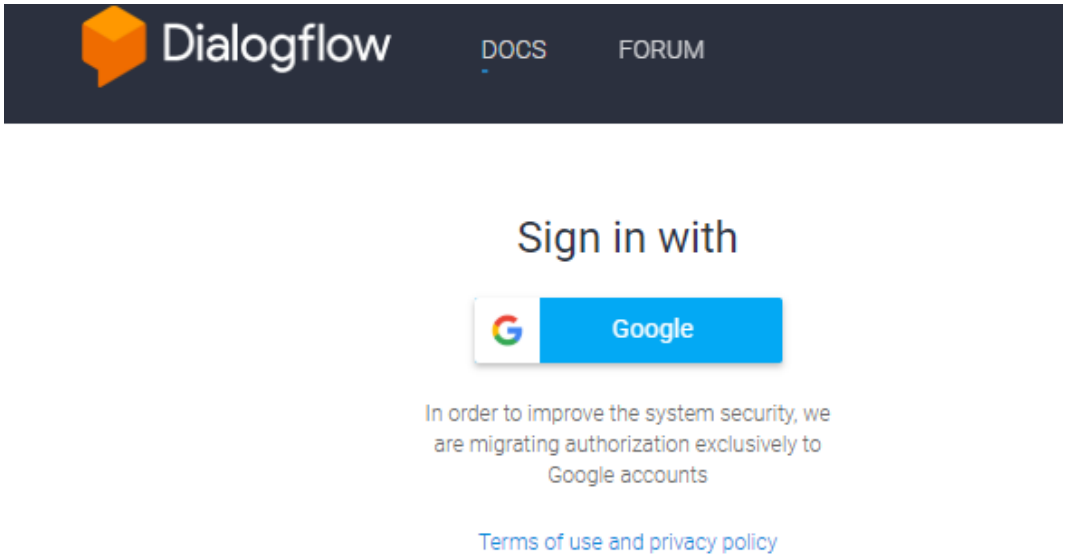
```
42.             JSONObject correction = correctionArray.getJSONObject(i);
43.
44.             message = correction.getString(LT_MESSAGE);
45.
46.             JSONArray replacementArray = correction.getJSONArray(LT_REPL
    ACEMENTS);
47.
48.             replacements = new String[replacementArray.length()];
49.
50.
51.             offset = correction.getInt(LT_OFFSET);
52.             length = correction.getInt(LT_LENGTH);
53.             sentence = correction.getString(LT_SENTENCE);
54.             ruleId = correction.getJSONObject(LT_RULE).getString(LT_RULE
    _ID);
55.             try {
56.                 ruleSubId = correction.getJSONObject(LT_RULE).getInt(LT_
    RULE_SUB_ID);
57.             }catch (Exception e){
58.                 ruleSubId=0;
59.
60.             }
61.             ruleDesc = correction.getJSONObject(LT_RULE).getString(LT_RU
    LE_DESC);
62.
63.             Correction correction = new Correction(messageId,message,off
    set,length,sentence,ruleId,ruleSubId,ruleDesc);
64.             correction.setId(correctionId);
65.
66.             for(int j=0;j<replacementArray.length();j++){
67.                 //what to do with replacementArray
68.             }
69.
70.             corrections[i] = correction;
71.         }
72.
73.         return corrections;
74.     }
75. }
```

Langkah terakhir ialah panggil fungsi getSimpleDataFromJson() yang baru saja dibuat. Fungsi ini dipanggil setelah fungsi getResponseFromHttpUrl().

```
1.  try {
2.      languagetoolJsonResponse = getResponseFromHttpUrl(searchUrl);
3.
4.      corrections = LanguagetoolResponseParser.getSimpleDataFromJson((Cont
    ext)listener, languagetoolJsonResponse, messageId);
5.
6.  } catch (IOException e) {
```

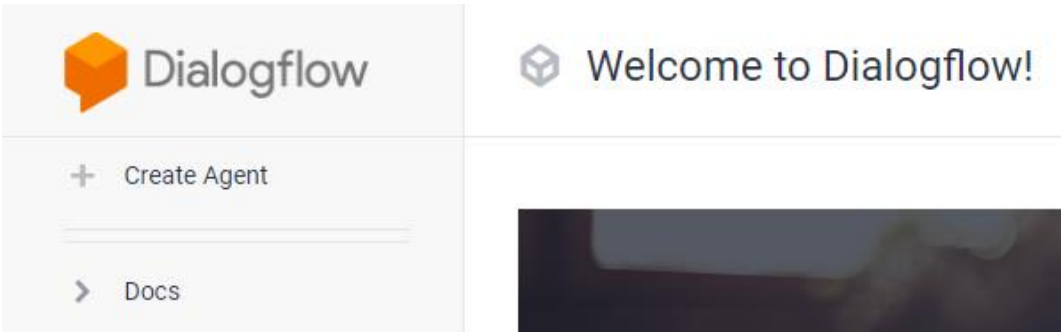
4. Dialogflow API

Untuk dapat memanfaatkan fitur API *Dialogflow*, developer diharuskan membuat *Agent* terlebih dahulu melalui *dialogflow developer console*. *Agent* ini yang nantinya akan dilatih atau di *update knowledge base* nya agar dapat merespon setiap pesan *user* dengan benar. Semakin banyak dan beragam masukan terhadap *knowledge base* nya maka semakin pintar *agent/bot* nya nanti menjawab pesan *user*. Sebelum ke pembuatan agent, pertamakali yang harus developer lakukan ialah *login/sign up* dengan menggunakan akun *Gmail*. Lihat gambar 3.13.



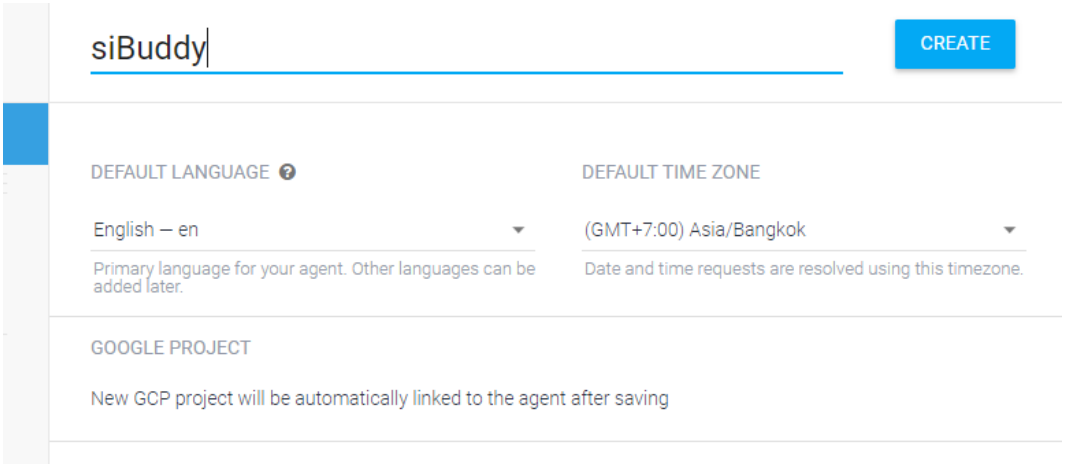
Gambar 3.13 Halaman *Sign in Dialogflow Developer Console*

Setelah *login* maka *user* akan masuk ke halaman *developer console*. Untuk membuat *agent* baru maka pengguna dapat memilih *Create Agent* di sisi kiri atas. Lihat gambar 3.14.



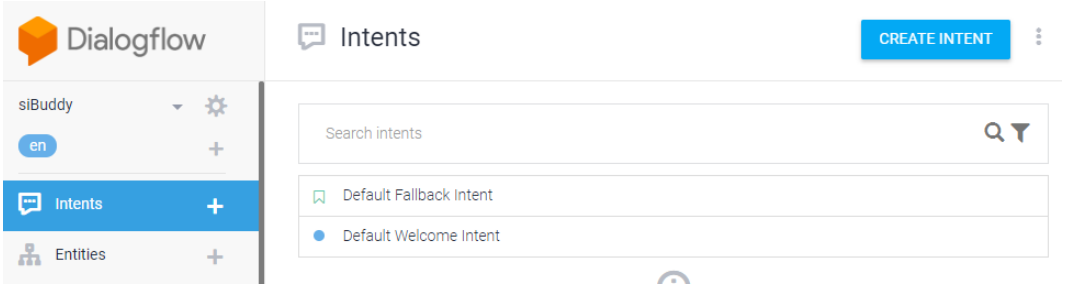
Gambar 3.14 *Create Agent*

Selanjutnya developer melakukan konfigurasi pada *Agent* yang akan dibuat. Konfigurasi itu adalah nama *agent*, bahasa, dan *time zone*. Lihat gambar 3.15.



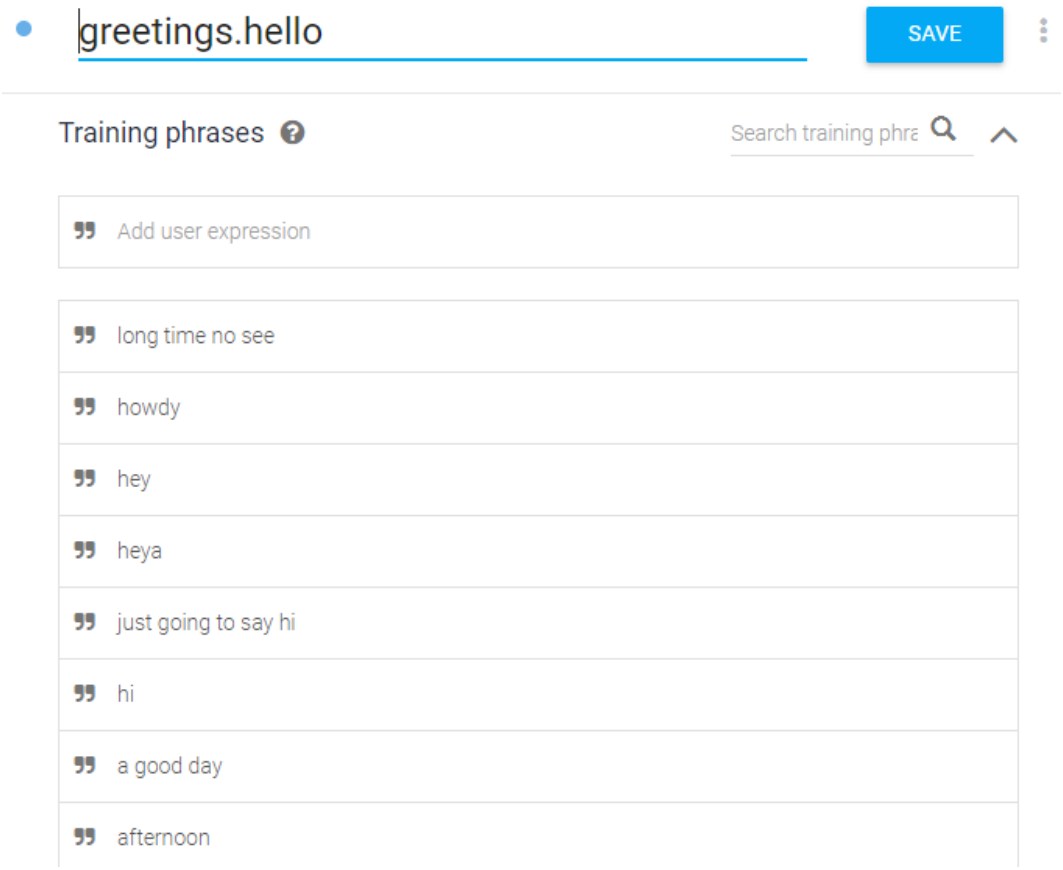
Gambar 3.15 Konfigurasi *Agent*

Selesai *agent* dibuat, maka developer harus menambahkan *knowledge base* kepada *agent*. *Knowledge base* itu berupa *Intent*. Maka langkah selanjutnya developer membuat *Intent*. Pilih menu *Intent* di sisi kiri dan pilih menu *CREATE INTENT*. Lihat gambar 3.16.

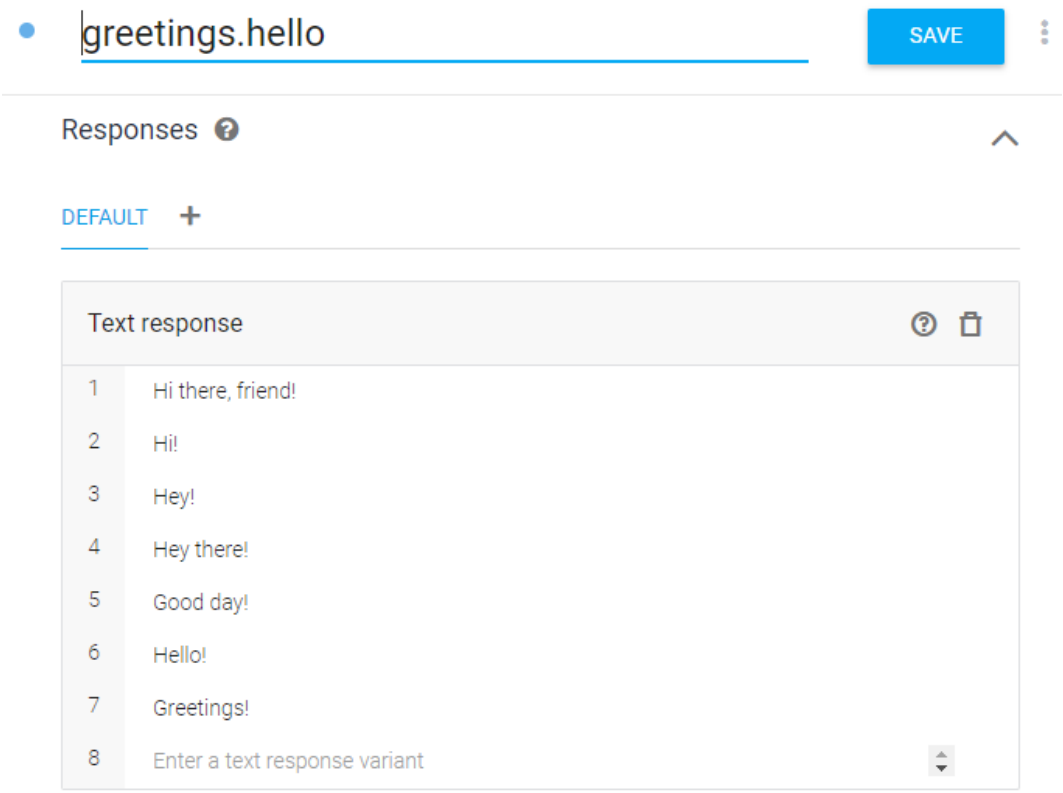


Gambar 3.16 *Create Intent*

Berikut ini adalah contoh pembuatan *Intent*, misalnya *intent* dengan nama greetings.hello. Lihat gambar 3.17 dan gambar 3.18.



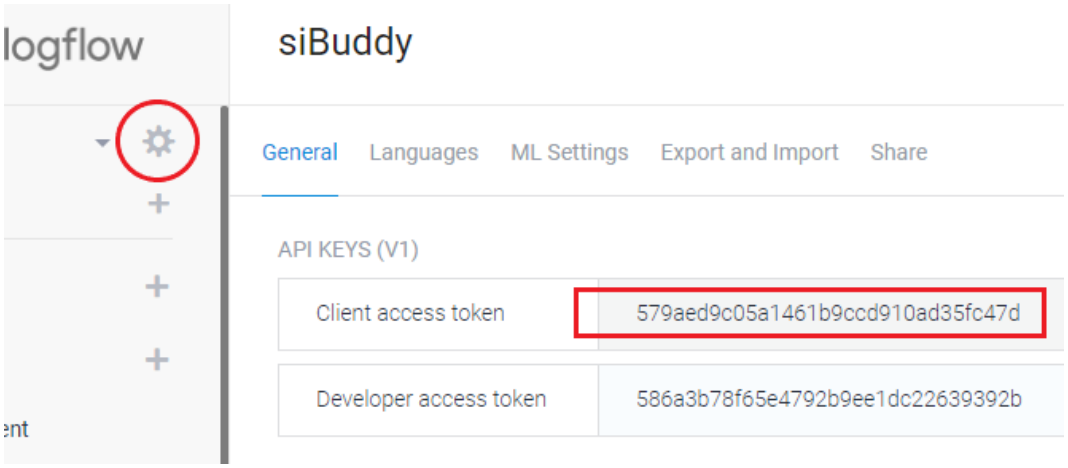
Gambar 3.17 Kolom *Training Phrase*



Gambar 3.18 Kolom *Responses*

Di kolom *Training Phrase*(Gambar 3.8) adalah semua kemungkinan frasa yang *user* gunakan ketika menyapa *bot*. Sedangkan di kolom *Responses* (gambar 3.9) adalah beberapa respon dari bot. Data di *training phrase* kemudian di *training* agar dapat cocok dengan frasa serupa tetapi tidak persis sama. Oleh sebab itu semakin banyak data di *training phrase* maka hasil yang diperoleh semakin optimal. Jika *request user* cocok dengan *intent* greetings.hello, maka respon akan diberikan secara acak.

Setelah *agent* siap, maka langkah selanjutnya ialah implementasi di aplikasi. Namun sebelum itu, diperlukan *client access token* yang diperlukan untuk inisialisasi di kode. *Client access token* dapat diakses dengan memilih icon *setting* > *General* > *Client access token*. Lihat gambar 3.19.



Gambar 3.19 *Client Access Token*

Untuk implementasi di program, *dialogflow* telah menyediakan kelas-kelas untuk mempermudah implementasi. Kelas tersebut yakni AIConfiguration, AIDataService, dan AIResult. Langkah pertama ialah membuat kelas yang akan menangani *request* ke *dialogflow*, kelas itu adalah DialogflowRequestHandler. Sama seperti *languagetool*, proses ini akan membutuhkan waktu sehingga proses ini akan dijalankan di background. Oleh karena itu, kelas ini akan mewariskan kelas Asynctask.

```
1.  public class DialogflowRequestHandler extends AsyncTask<AIRequest, Void,
     AIResponse> {
2.
3.      public DialogflowRequestHandler() {
4.
5.      }
6.
7.      @Override
8.      protected AIResponse doInBackground(AIRequest... requests) {
9.
10.     }
11.
12.     @Override
13.     protected void onPostExecute(AIResponse aiResponse) {
14.
15.     }
16. }
```

Langkah selanjutnya adalah inisialisai objek AIDataService di konstruktor.

```
final AIConfiguration config = new AIConfiguration(CLIENT_ACCESS_TOKEN,
                                                   AIConfiguration.SupportedLanguages.English,
                                                   AIConfiguration.RecognitionEngine.System);

AIDataService aiDataService = new AIDataService(config);
```

Setelah itu, buat fungsional untuk *request reply* di doInBackground. Proses ini akan mengembalikan respon bertipe AIResponse.

```
@Override
protected AIResponse doInBackground(AIRequest... requests) {
    final AIRequest request = requests[0];

    try {
        final AIResponse response = aiDataService.request(request);
        return response;
    } catch (AIServiceException e) {

    }
    return null;
}
```

Buat *instance* dari kelas DialogflowRequestHandler dan panggil fungsi execute() di Activity bersangkutan.

```
DialogflowRequestHandler dialogflowHandler = new DialogflowRequestHandler();
dialogflowHandler.execute(aiRequest);
```

Adapun untuk mendapatkan respon sederhana, ditangani oleh fungsi *static* getSimpleData dari kelas DialogflowResponseParser.

```
final AIResponse response = aiDataService.request(AIRequest);

String reply = response.getResult().getFulfillment().getSpeech();
String action = response.getResutl().getAction();
```

3.1.6. Analisis Data

Sebagaimana telah dijelaskan pada batasan masalah sebelumnya, *chatbot* mampu membuat percakapan dengan pengguna menggunakan Bahasa Inggris. Adapun percakapan yang diambil untuk dijadikan *knowledge-base* berasal dari [7], yang memuat 37 topik percakapan, ditunjukan pada tabel 3.1.

Tabel 3.1 37 Topik Percakapan

No	Topik
1	*Greetting and parting*
2	*Asking name and spelling*
3	*Talking about origin and nationality*
4	*Introducing youself and other people*
5	*Asking job and work places*
6	*Asking age, height, weight, and marital status*
7	*Asking address, location, and telephone number*
8	*Talking about family, appearances, reseamblance, and personality*
9	*Asking people's hobbies, likes and dislikes*
10	*Asking times and schedules*
11	*Talking about time, routines, and frequency*
12	*Identifying things and talking about their position*
13	*Renting a house*
14	*Shopping*
15	*Describing people's clothes*
16	*Healt and giving suggestions*
17	*Means of transportation*
18	*Talking about past activities*
19	*Talking about abilities*
20	*Comparing people and things*
21	*What is he doing at the moment?*
22	*What are ou doing tomorrow?*
23	*What date it is today?*
24	*Addition, substraction, division, and multiplication*
25	*Giving instructions*
26	*Lending and borrowing*
27	*Eating in a restaurant*
28	*Asking for and giving permission*
29	*Offering to help*
30	*Making a request*
31	*Invite someone out*
32	*Making a phone call*
33	*Asking and giving opinions, expressing agreement and disagreement*
34	*Talking about possessions*
35	*Talking about past habits*
36	*Talking about changes*
37	*Saving number and dates*

3.2. Analisis Kebutuhan Non Fungsional

Kebutuhan non fungsional adalah usulan yang direkomendasikan kepad pengguna agar perangkat lunak yang akan dibangun menjadi *user friendly* dan perangkat keras yang mendukung secara maksimal terhadap kinerja perangkat lunak.

3.2.1. Kebutuhan Perangkat Lunak

Spesifikasi perangkat lunak yang dibutuhkan agar sistem dapat berjalan dengan baik dapat dilihat pada tabel 3.2.

Tabel 3.2 Kebutuhan Perangkat Lunak

Sistem operasi	OS Andoid 4.0.3 (*Ice cream sandwich*)

3.2.2. Kebutuhan Perangkat Keras

Perangkat keras adalah seluruh komponen atau unsur peralatan yang digunakan untuk menunjang pembangunan aplikasi. Adapun perangkat keras yang digunakan untuk pemakaian aplikasi ini secara optimal dapat dilihat pada tabel 3.3.

Tabel 3.3 Kebutuhan Perangkat Keras

Smartphone	Android 4.0.3 (*Ice cream sandiwich*)
Ram	RAM 512MB
Koneksi data	Data Koneksi HSDPA/EVDO

3.3. Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional dapat didefinisikan sebagai penggambaran, perencanaan dan pembuatan sketsa atau pengaturan dari beberapa elemen yang terpisah ke dalam satu kesatuan yang utuh dan berfungsi. Alat bantu yang digunakan untuk menggambarkan sistem secara umum yang akan dibangun yaitu dengan pendekatan berorientasi objek.

3.3.1. Identifikasi Aktor

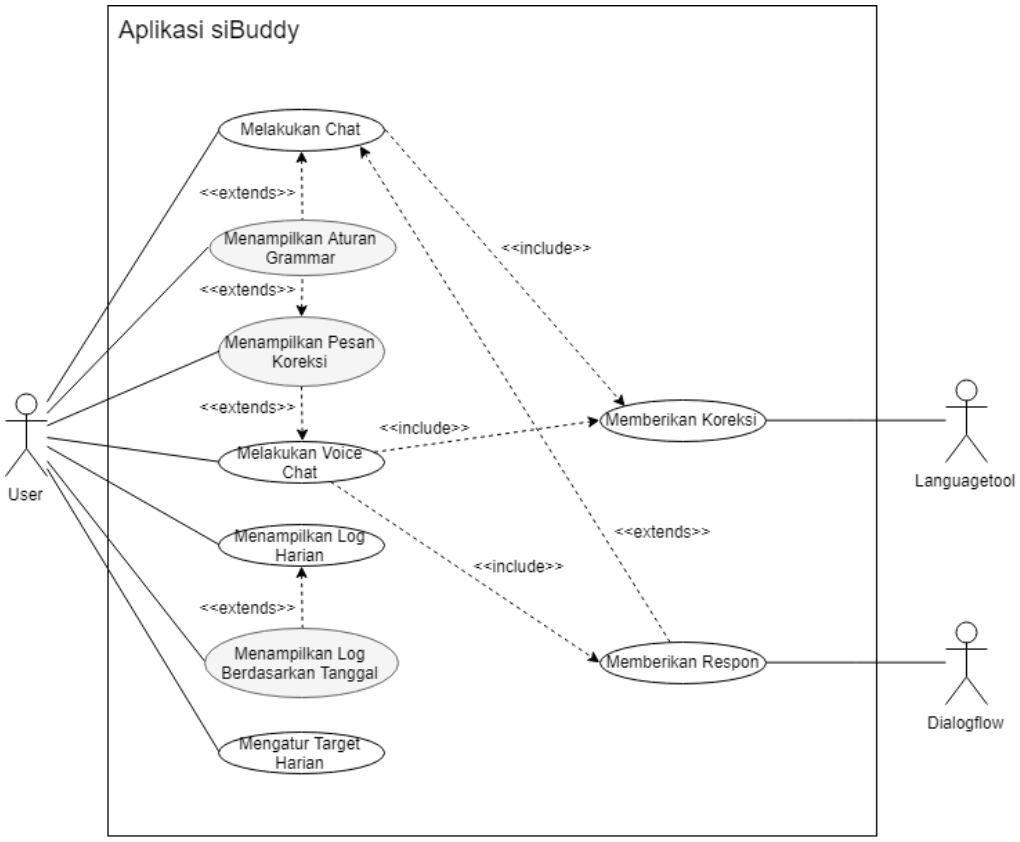
Tahap pertama yang dilakukan dalam melakukan analisis berorientasi objek menggunakan UML adalah menentukan aktor atau pengguna dari sistem. Identifikasi aktor dimaksudkan untuk mengetahui siapa saja aktor yang terlibat di dalam sistem aplikasi ini *chatbot* ini. Deskripsi dari tiap aktor yang terlibat dapat dilihat pada tabel 3.4.

Tabel 3.4 Identifikasi Aktor

Aktor	**Deskripsi**
Pengguna	Merupakan seseorang yang menginstall aplikasi dan menggunakannya.
Dialogflow	Merupakan API yang digunakan untuk merespon percakapan user
Languagetool	Merupakan API yang digunakan untuk mengoreksi kesalahan kalimat.

3.3.2. Use Case Diagram

Diagram *use case* memperlihatkan hubungan-hubungan yang terjadi antara aktor-aktor dengan *use case* dalam sistem. Salah satu manfaat dari diagram *use case* adalah untuk komunikasi. Calon pengguna sistem/perangkat lunak dapat mengamati diagram *use case* untuk mendapatkan pemahaman yang utuh tentang sistem yang akan dikembangkan. *Use case diagram* sistem dapat dilihat pada gambar 3.20.



Gambar 3.20 Diagram *Use Case*

3.3.3. Definisi Use Case

Definisi *use case* berfungsi untuk menjelaskan proses yang terdapat pada setiap *use case*. Definisi *usecase* diagram akan dijelaskan pada tabel 3.5.

Tabel 3.5 Definisi Use Casee

No	Use case	Deskripsi
1	Melakukan Chat	Proses membuat percakapan dengan bot via teks
2	Melakukan Voice Chat	Proses membuat percakapan dengan bot via suara
3	Memberikan Koreksi	Proses memberikan respon berupa koreksi terhadap request yang diterima
4	Memberikan Respon	Proses memberikan respon balasan terhadap request yang diterima
5	Menampilkan Pesan Koreksi	Proses menampilkan halaman koreksi pada mode voice chat
6	Menampilkan Aturan Grammar	Proses menampilkan informasi lebih lanjut terkait kesalahan grammar dengan membuka web page
7	Menampilkan Log Harian	Proses melihat log aktivitas harian untuk evaluasi
8	Menampilkan Log Berdasarkan Tanggal	Proses melakukan filter terhadap log harian berdasarkan tanggal
9	Mengatur Target Harian	Proses menetapkan target harian yang ingin dicapai

3.3.4. Skenario Use Case Diagram

Skenario *Use Case* mendeskripsikan urutan langkah-langkah dalam aplikasi, baik yang dilakukan aktor terhadap sistem maupun yang dilakukan oleh sistem terhadap aktor. Berikut adalah skenario dari masing-masing *use case* :

1. *Use Case Scenario* Melakukan *Chat*

   Use Case Scenario Melakukan *Chat* dapat dilihat pada tabel 3.6

Tabel 3.6 Skenario *use case* Melakukan *Chat*

Use Case Name	Melakukan *Chat*	
Related Requirement	Berada di halaman *chat*	
Goal in Content	Membuat percakapan teks dengan *bot*	
Precondition	Sistem menampilkan chat terakhir	
Successful End Condition	Sistem membalas pesan *user* dan melakukan update terhadap halaman *chat*	
Failed End Condition		
Primary Actor	*User*	
Secondary Actor	*System*	
Trigger		
Included Class	ChatActivity, Message, DBHelper, ChatAdapter, LanguagetoolRequestHandler, NetworkUtil, AIRequest, DialogflowRequestHandler	
Main Flow	*Step*	*Action*
	1	*User* mengetikkan pesan dan menekan tombol kirim
	2	Sistem menyimpan pesan *User* ke

		database
	3	Sistem melakukan *request* koreksi ke *Languagetool*
	4	Sistem mendapatkan respon koreksi dari languagetool
	5	Sistem menyimpan data koreksi, *replacement*, dan pesan koreksi ke database
	6	Sistem membaca pesan dari *database*
	7	Sistem memperbarui daftar *chat* terakhir
Extention	*Step*	*Branching Action*
	4.1	Sistem meneruskan request ke Dialogflow
	4.2	Sistem mendapatkan respon dari *dialogflow*
	4.3	Sistem menyimpan pesan balasan ke *database*
	4.4	Sistem membaca pesan dari *database*
	4.5	Sistem memperbarui daftar *chat* terakhir

2. *Use Case Scenario* Melakukan *Voice Chat*

Use Case Scenario Melakukan *Voice Chat* dapat dilihat pada tabel 3.7

Tabel 3.7 Skenario *use case* Melakukan *Voice Chat*

Use Case Name	Melakukan *Voice Chat*	
Related Requirement	Berada di halaman *voice chat*	
Goal in Content	Membuat percakapan suara dengan *bot*	
Precondition	Tombol *speak* dalam keadaan *enable*	
Successful End Condition	Sistem merespon *request user* dengan suara	
Failed End Condition		
Primary Actor	*User*	
Secondary Actor	*System, Languagetool, Dialogflow*	
Trigger		
Included Class	VoiceChatFragment, TTSHandler, LanguagetoolRequestHandler, NetworkUtil, VoiceChatActivity, AIRequest, DialogflowRequestHandler	
Main Flow	*Step*	*Action*
	1	*User* Menekan tombol *Speak* dan memberikan pesan suara
	2	Sistem melakukan *speech recognition* terhadap pesan suara
	3	Sistem menyimpan pesan(teks) ke *database*
	4	Sistem melakukan *request* koreksi ke *languagetool*
	5	Sistem mendapatkan respon koreksi dari *languagetool*
	6	Sistem menyimpan data koreksi dan *replacement* ke *database*, sedangkan pesan koreksi disimpan ke sebuah *arraylist*

	7	Sistem meneruskan *request* ke *Dialogflow*
	8	Sistem mendapatkan respon dari *dialogflow*
	9	Sistem melakukan konversi teks-suara (*text-to-speech*)
	10	Sistem memutar suara sebagai pesan balasan
Extension	*Step*	*Branching Action*
	5.1	Sistem meneruskan *request* ke *Dialogflow*
	5.2	Sistem mendapatkan respon dari *dialogflow*
	5.3	Sistem melakukan konversi teks-suara (*text-to-speech*)
	5.4	Sistem memutar suara sebagai pesan balasan

3. *Use Case Scenario* Memberikan Koreksi

Use Case Scenario Memberikan Koreksi dapat dilihat pada tabel 3.8

Tabel 3.8 Skenario *Use Case* Memberikan Koreksi

Use Case Name	Memberikan Koreksi	
Related Requirement	Parameter url *request*	
Goal in Content	Mendapatkan respon koreksi dari *languagetool*	
Precondition	Sistem melakukan *request* melalui objek *handler*	
Successful End Condition	Sistem memberikan respon koreksi dalam format sederhana	
Failed End Condition		
Primary Actor	*System*	
Secondary Actor	*Languagetool*	
Trigger		
Included Class	LanguagetoolRequestHandler, NetworkUtil, LanguagetoolResponseparser, DBHelper, Correction, Replacement, ChatActivity	
Main Flow	*Step*	*Action*
	1	Sistem *request* koreksi ke *languagetool*
	2	*Languagetool* memberikan respon koreksi
	3	Sistem melakukan *parsing* terhadap respon koreksi
Extention	*Step*	*Branching Action*

4. *Use Case Scenario* Memberikan Respon

   Use Case Scenario Memberikan Respon dapat dilihat pada tabel 3.9

Tabel 3.9 Skenario *Use Case* Memberikan Respon

Use Case Name	Memberikan Respon	
Related Requirement	Parameter pesan *request*	
Goal in Content	Mendapatkan respon balasan dari *dialogflow*	
Precondition	Sistem melakukan *request* melalui objek *handler*	
Successful End Condition	Sistem mendapatkan respons *dialogflow* dalam format sederhana	
Failed End Condition		
Primary Actor	*System*	
Secondary Actor	*Dialogflow*	
Trigger		
Included Class	DialogflowRequestHandler, AIDataService, DialogflowResponseParser, AIResponse, Message, ChatActivity	
Main Flow	*Step*	*Action*
	1	Sistem melakukan *request* balasan ke *dialogflow*
	2	*Dialogflow* memberikan respon
	3	Sistem melakukan penguraian respon kedalam format sederhana
Extentions	Step	Branching Action

5. *Use Case Scenario* Menampilkan Pesan Koreksi

   Use Case Scenario Menampilkan Pesan Koreksi dapat dilihat pada tabel 3.10.

Tabel 3.10 Skenario *Use Case* Menampilkan Pesan Koreksi

Use Case Name	Menampilkan Pesan Koreksi	
Related Requirement	Berada di halaman *voice chat*	
Goal in Content	Melihat daftar koreksi percakapan suara	
Precondition		
Successful End Condition	Sistem menampilkan daftar koreksi percakapan suara	
Failed End Condition	Belum ada data	
Primary Actor	*User*	
Secondary Actor	*System*	
Trigger		
Included Class	VoiceChatFragment, VoiceChatActivity, CorrectionMessageFragment	
Main Flow	*Step*	*Action*
	1	*User* Menekan icon "*Bell*" pada halaman *voice chat*
	2	Sistem menampilkan daftar koreksi yang sebelumnya disimpan dalam *arraylist*
Extension	*Step*	*Branching Action*
	2.1	Sistem menampilkan pesan "Tidak ada

		koreksi”

6. *Use Case Scenario* Menampilkan Aturan *Grammar*

   Use Case Scenario Menampilkan Aturan *Grammar* dapat dilihat pada tabel 3.11.

Tabel 3.11 Skenario *Use Case* Menampilkan Aturan *Grammar*

Use Case Name	Menampilkan Aturan *Grammar*	
Related Requirement	Terdapat pesan koreksi	
Goal in Content	Melihat informasi lebih lanjut mengenai koreksi	
Precondition		
Successful End Condition	Sistem menampilkan informasi lebih lanjut tentang koreksi dengan membuka halaman *web*	
Failed End Condition		
Primary Actor	*User*	
Secondary Actor	*System*	
Trigger		
Included Class	ChatAdapter, ChatActivity, GrammarRuleActivity	
Main Flow	*Step*	*Action*
	1	*User* Menekan pesan koreksi
	2	Sistem membuka halaman *web* dan menampilkan informasi aturan *grammar* terkait
Extension	*Step*	*Branching Action*

7. *Use Case Scenario* Menampilkan Log Harian

   Use Case Scenario Menampilkan Log Harian dapat dilihat pada tabel 3.12.

Tabel 3.12 Skenario *Use Case* Menampilkan Log Harian

Use Case Name	Menampilkan Log Harian	
Related Requirement	Berada di halaman *chat*	
Goal in Content	Melihat log harian	
Precondition		
Successful End Condition	Sistem menampilkan log harian	
Failed End Condition		
Primary Actor	*User*	
Secondary Actor	*System*	
Trigger		
Included Class	DailyLogActivity, DBHelper	
Main Flow	*Step*	*Action*
	1	*User* menekan *icon* “*Star*”
	2	Sistem memuat data log harian dari *database* untuk hari sekarang
	3	Sistem menampilkan log harian dalam bentuk angka dan grafik
Extension	*Step*	*Branching Action*

8. *Use Case Scenario* Menampilkan Log Berdasarkan Tanggal

   Use Case Scenario Menampilkan Log Berdasarkan Tanggal dapat dilihat pada tabel 3.13

Tabel 3.13 Skenario *Use Case* Menampilkan Log Berdasarkan Tanggal

Use Case Name	Menampilkan Log Berdasarkan Tanggal	
Related Requirement	Berada di halaman *Daily Log*	
Goal in Content	Melihat log harian untuk hari yang ditentukan	
Precondition		
Successful End Condition	Sistem menampilkan log harian berdasarkan hari yang ditentukan *user*	
Failed End Condition		
Primary Actor	*User*	
Secondary Actor	*System*	
Trigger		
Included Class	DailyLogActivity, DBHelper	
Main Flow	*Step*	*Action*
	1	*User* mengatur hari melalui kolom *date picker*
	2	Sistem memuat data log harian dari *database* berdasarkan tanggal yang dipilih *user*
	3	Sistem menampilkan log harian dalam bentuk angka dan grafik
Extension	*Step*	*Branching Action*

9. *Use Case Scenario* Mengatur Target Harian

   Use Case Scenario Mengatur Target Harian dapat dilihat pada tabel 3.14.

Tabel 3.14 Skenario *Use Case* Mengatur Target Harian

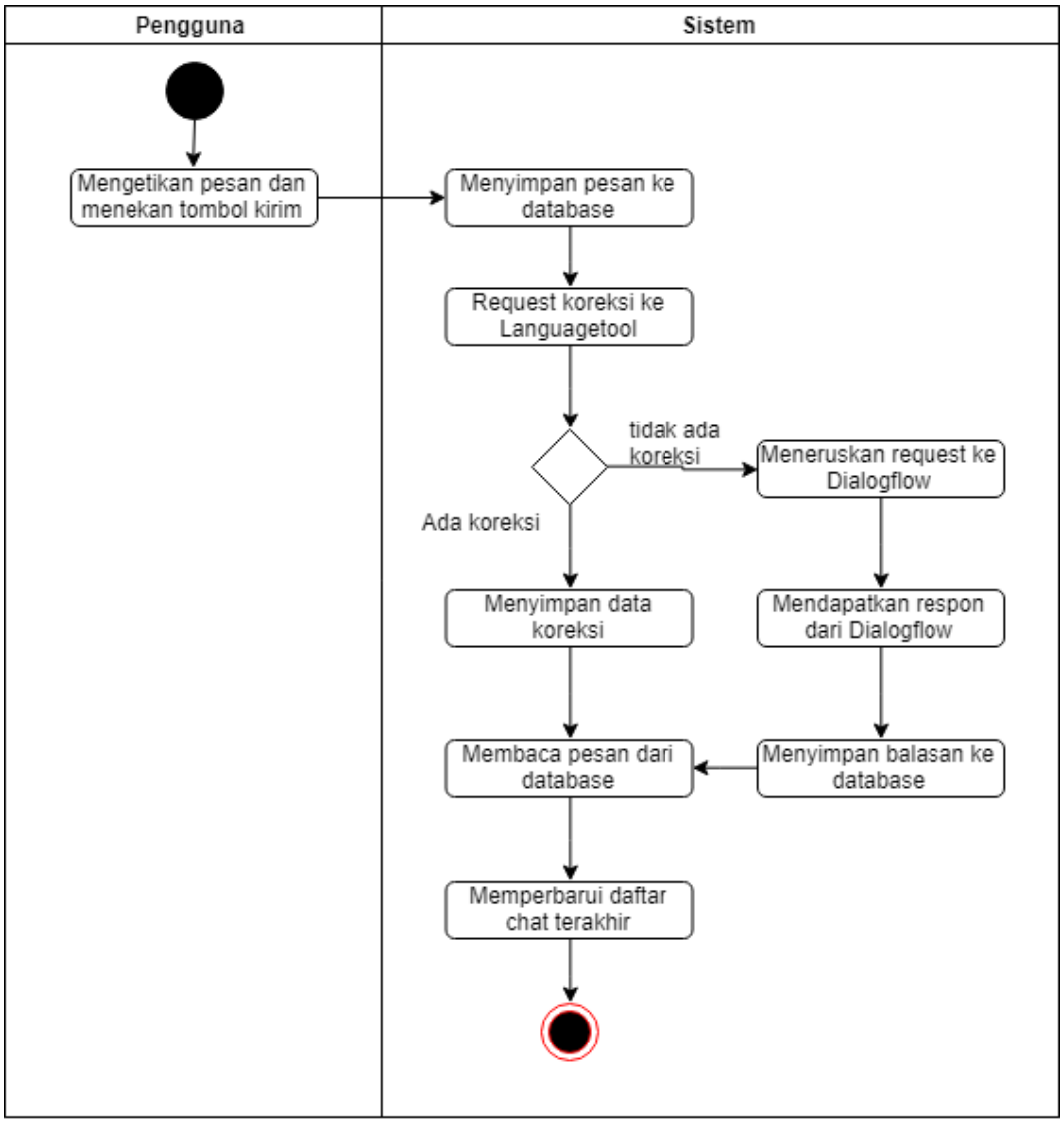
Use Case Name	Mengatur Target Harian	
Related Requirement	Berada di halaman *chat*	
Goal in Content	Mengatur target harian	
Precondition		
Successful End Condition	Sistem menyimpan pengaturan target harian	
Failed End Condition		
Primary Actor	*User*	
Secondary Actor	*System*	
Trigger		
Included Class	PreferencesActivity	
Main Flow	*Step*	*Action*
	1	*User* memilih menu "*settings*"
	2	Sistem menampilkan *form* target harian
	3	*User* mengatur nilai pada *form*
	4	Sistem menyimpan target harian
Extension	*Step*	*Branching Action*

3.3.5. Activity Diagram

Activity diagram merupakan diagram yang memodelkan aliran kerja atau *work flow* dari urutan aktifitas dalam suatu proses yang mengacu pada *use case* diagram yang ada. Berikut ini adalah *activity diagram* untuk setiap use case:

1. *Activity Diagram* Melakukan *Chat*

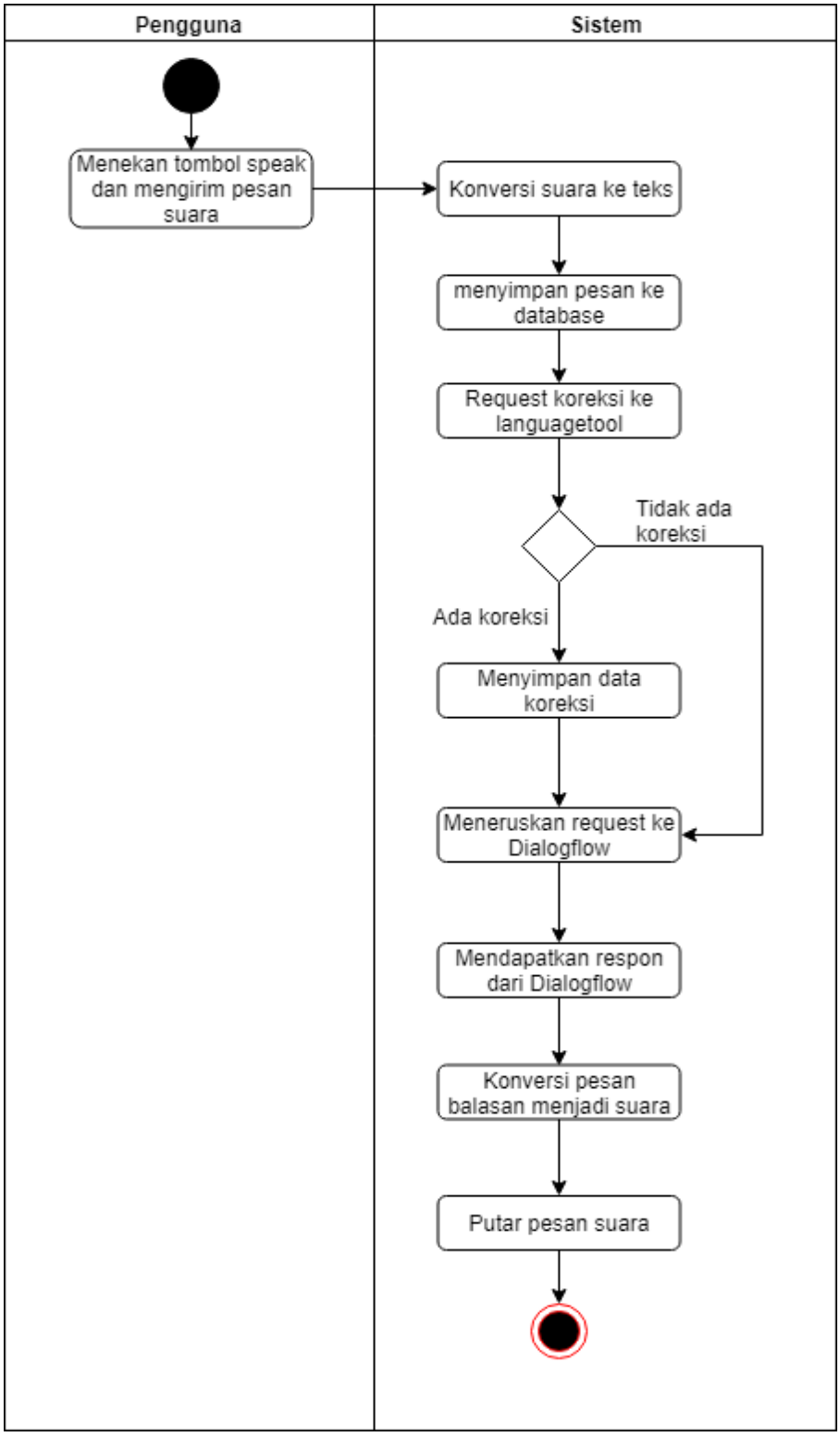
   Activity Diagram Melakukan *Chat* dapat dilihat pada gambar 3.21.



Gambar 3.21 Diagram *Activity* Melakukan *Chat*

2. *Activity Diagram* Melakukan *Voice Chat*

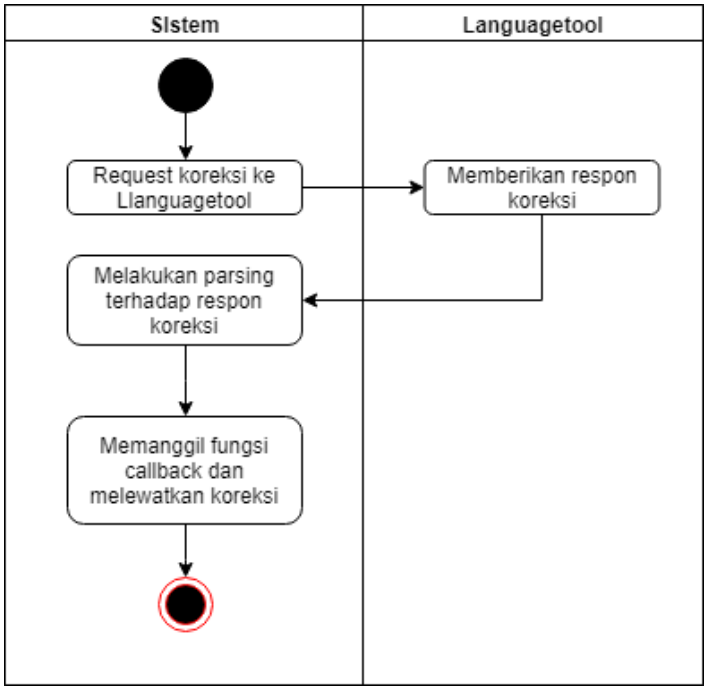
Activity Diagram Melakukan *Voice Chat* dapat dilihat pada gambar 3.22.



Gambar 3.22 Diagram *Activity* Melakukan Voice Chat

3. *Activity Diagram* Memberikan Koreksi

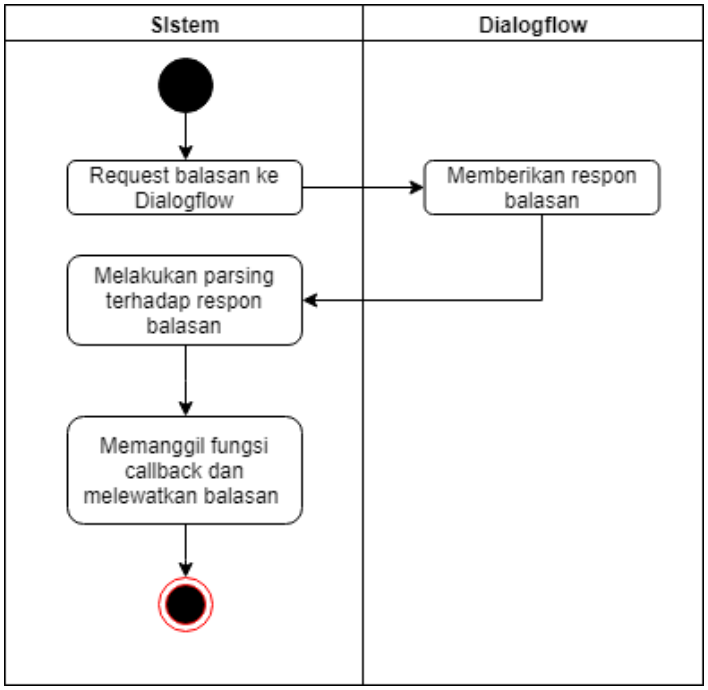
   Activity Diagram Memberikan Koreksi dapat dilihat pada gambar 3.23.



Gambar 3.23 Diagram *Activity* Memberikan Koreksi

4. *Activity diagram* Memberikan Respon

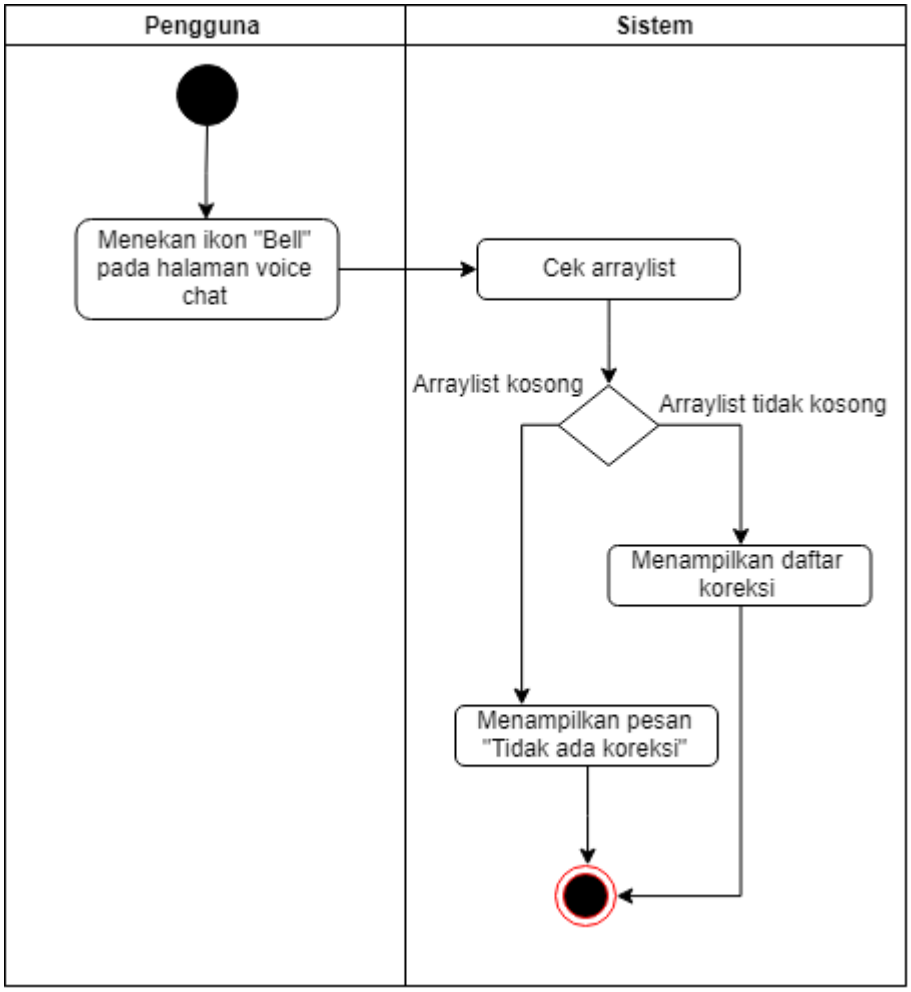
   Activity diagram Memberikan Respon dapat dilihat pada gambar 3.24.



Gambar 3.24 Diagram *Activity* Memberikan Respon

5. *Activity Diagram* Menampilkan Pesan Koreksi

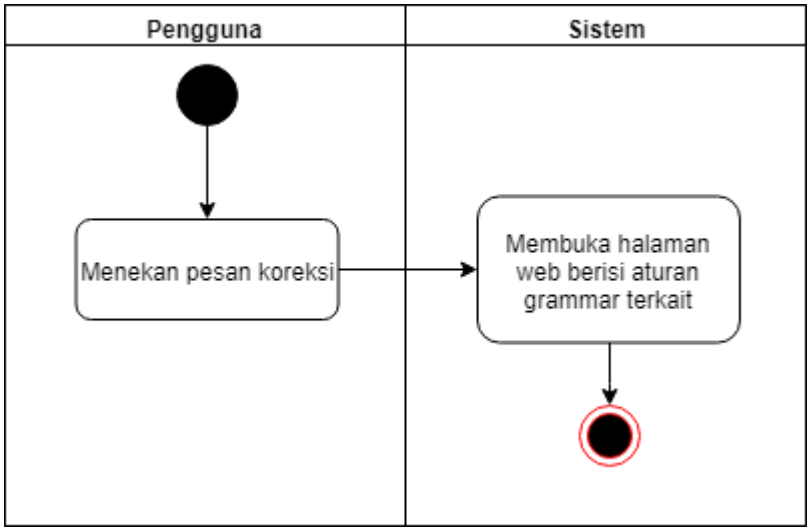
   Activity Diagram Menampilkan Pesan Koreksi dapat dilihat pada gambar 3.25.



Gambar 3.25 Diagram *Activity* Menampilkan Pesan Koreksi

6. *Activity Diagram* Menampilkan Aturan *Grammar*

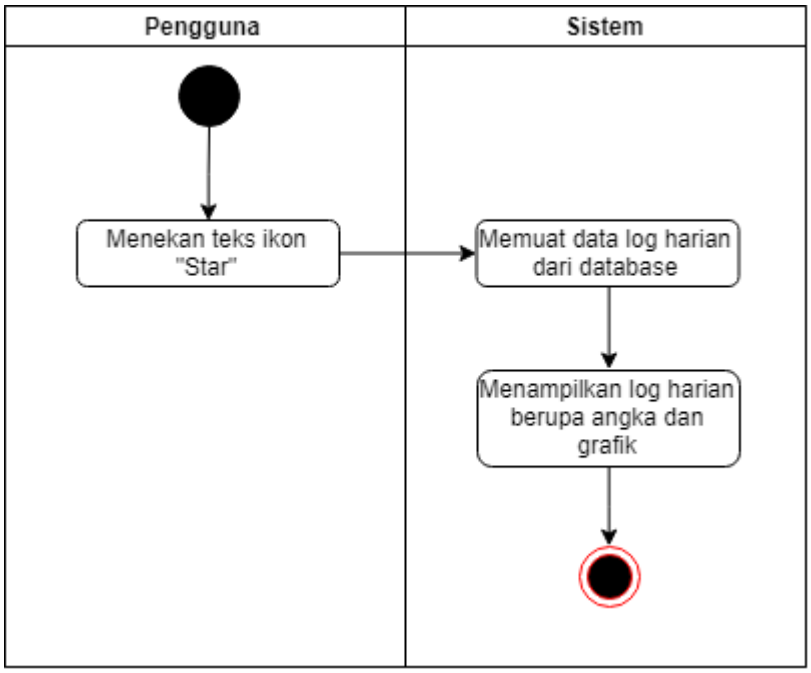
   Activity Diagram Menampilkan Aturan *Grammar* dapat dilihat pada gambar 3.26



Gambar 3.26 Diagram *Activity* Menampilkan Aturan Grammar

7. *Activity Diagram* Menampilkan Log Harian

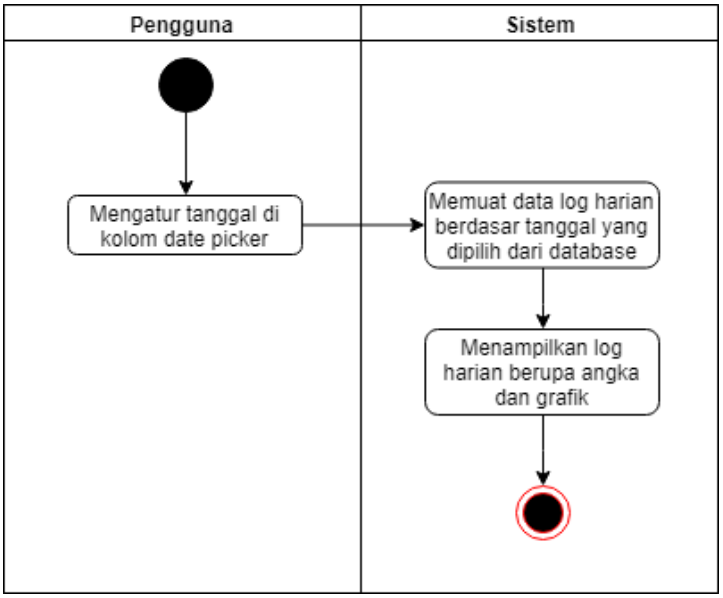
   Activity Diagram Menampilkan Log Harian dapat dilihat pada gambar 3.27.



Gambar 3.27 Diagram *Activity* Menampilkan Log Harian

8. *Activity Diagram* Menampilkan Log Berdasarkan Tanggal

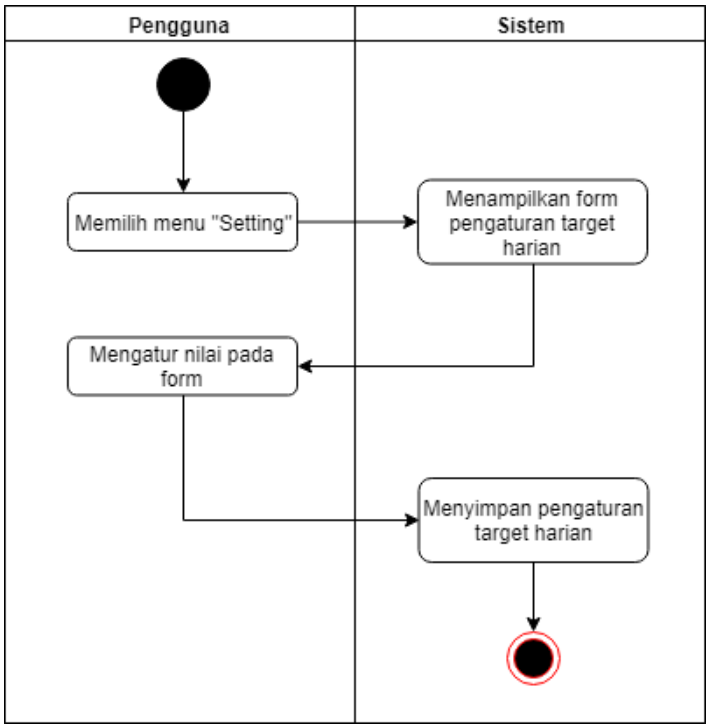
   Activity Diagram Menampilkan Log Berdasarkan Tanggal dapat dilihat pada gambar 3.28.



Gambar 3.28 Diagram *Activity* Menampilkan Log Berdasarkan Tanggal

9. *Activity Diagram* Mengatur Target Harian

   Activity Diagram Mengatur Target Harian dapat dilihat pada gambar 3.29.



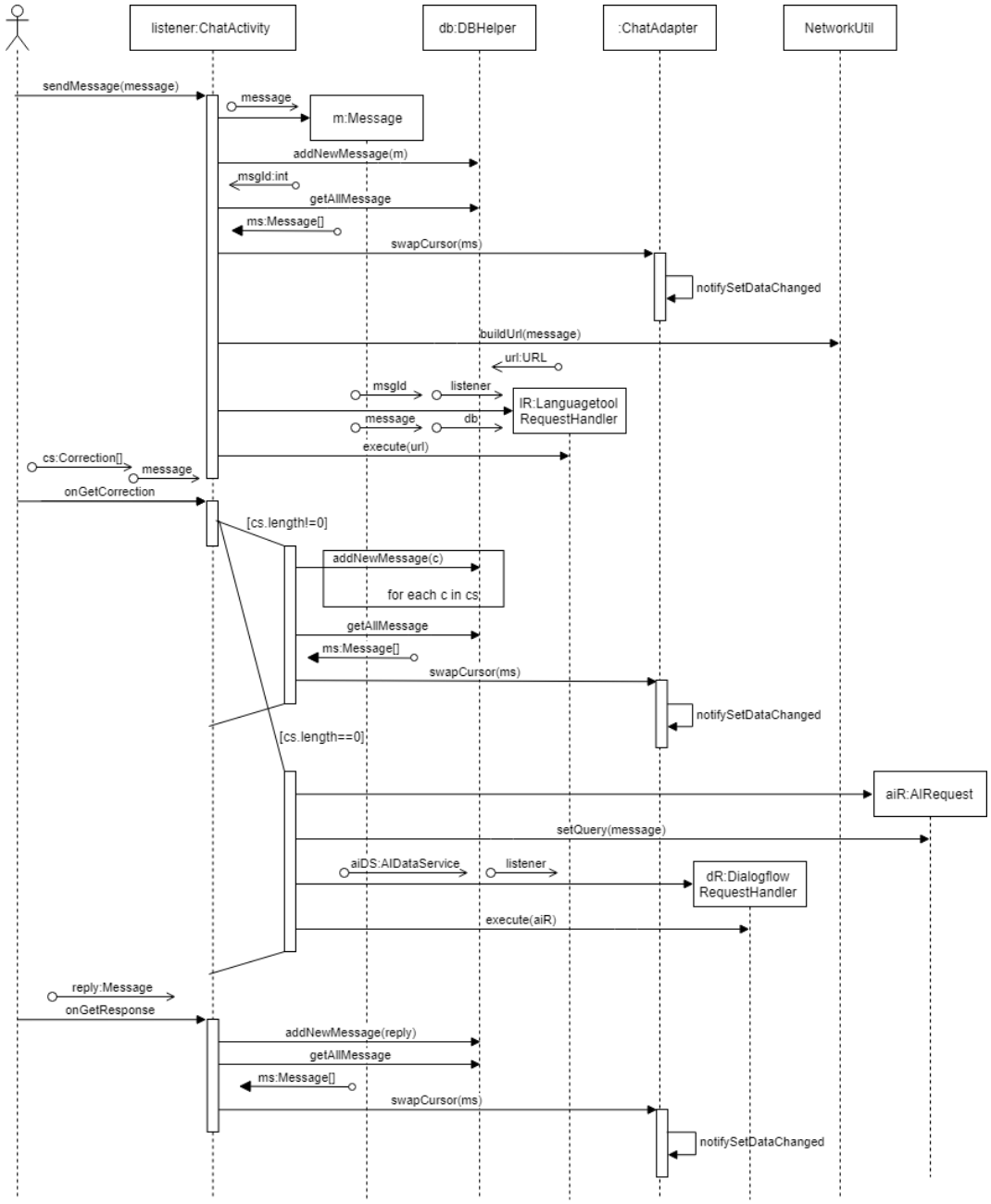
Gambar 3.29 Diagram *Activity* Mengatur Target harian

3.3.6. Sequence Diagram

Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara objek juga interaksi antara objek yang terjadi pada titik tertentu dalam eksekusi sistem. *Sequence diagram* sistem klasifikasi ini dibagi menjadi beberapa sub bagian:

1. *Sequence Diagram* Melakukan *Chat*

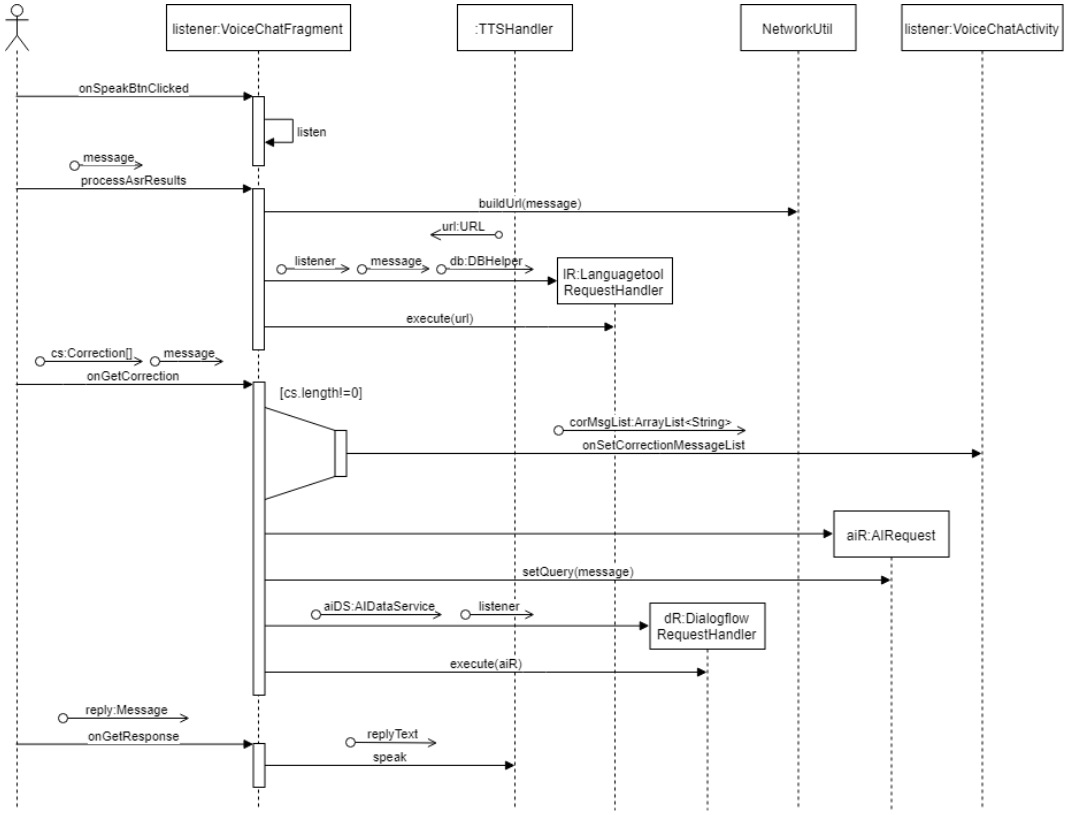
   Sequence Diagram Melakukan Chat dapat dilihat pada gambar 3.30.



Gambar 3.30 Diagram *Sequence* Melakukan *Chat*

2. *Sequence Diagram* Melakukan *Voice Chat*

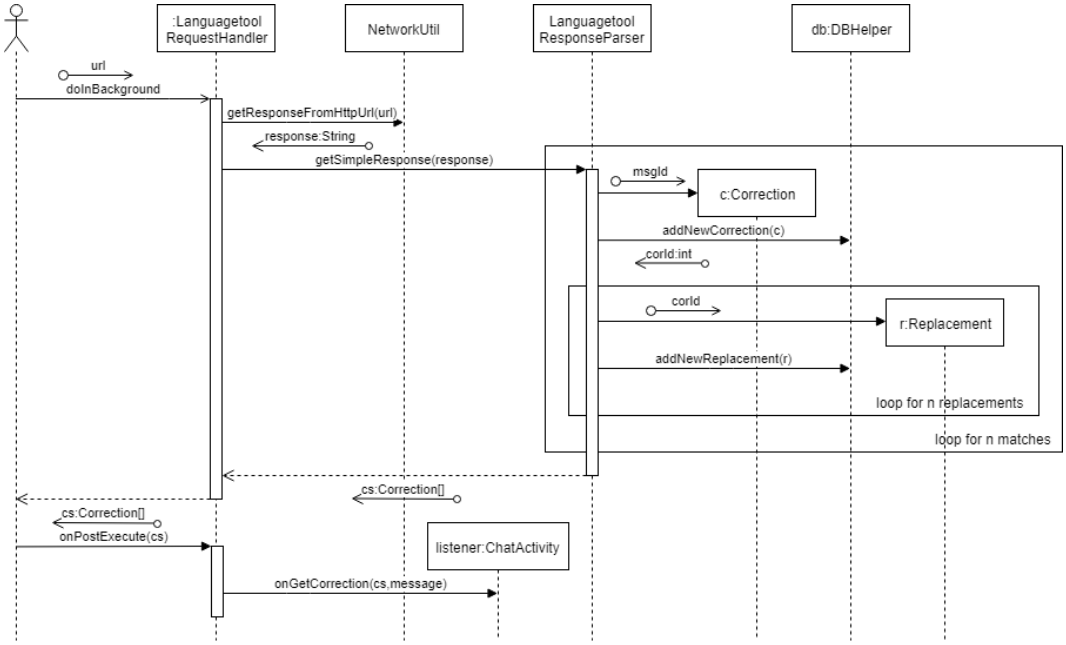
   Sequence Diagram Melakukan *Voice Chat* dapat dilihat pada gambar *3.31.*



Gambar 3.31 Diagram *Sequence* Melakukan *Voice Chat*

3. *Sequence Diagram* Memberikan Koreksi

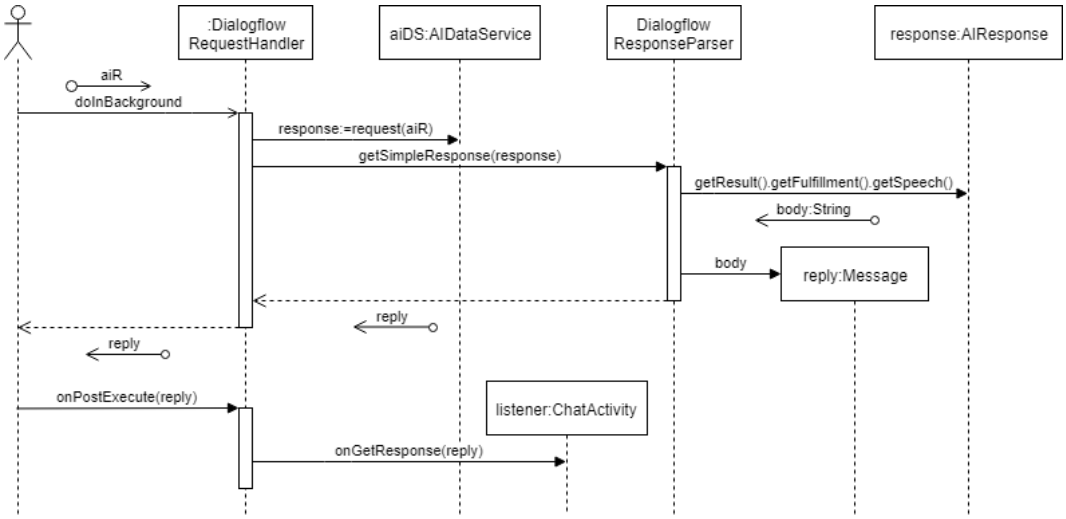
   Sequence Diagram Memberikan Koreksi dapat dilihat pada gambar 3.32.



Gambar 3.32 Diagram *Sequence* Memberikan Koreksi

4. *Sequence Diagram* Memberikan Respon

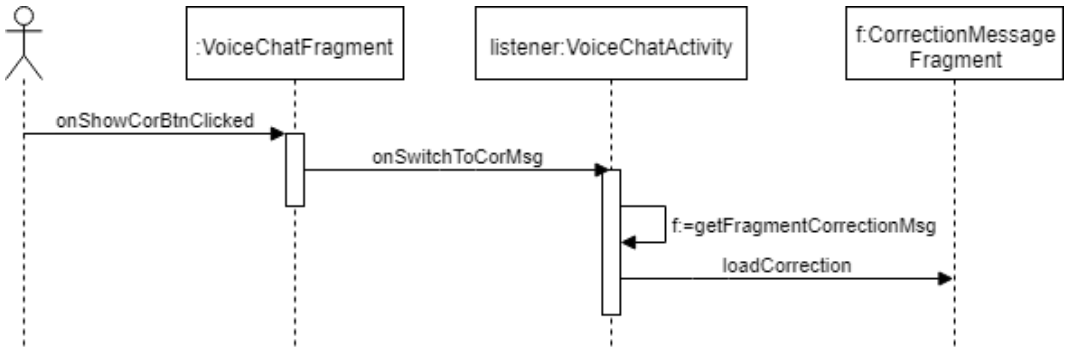
   Sequence Diagram Memberikan Respon dapat dilihat pada gambar 3.33.



Gambar 3.33 Diagram *Sequence* Memberikan Respon

5. *Sequence Diagram* Menampilkan Pesan Koreksi

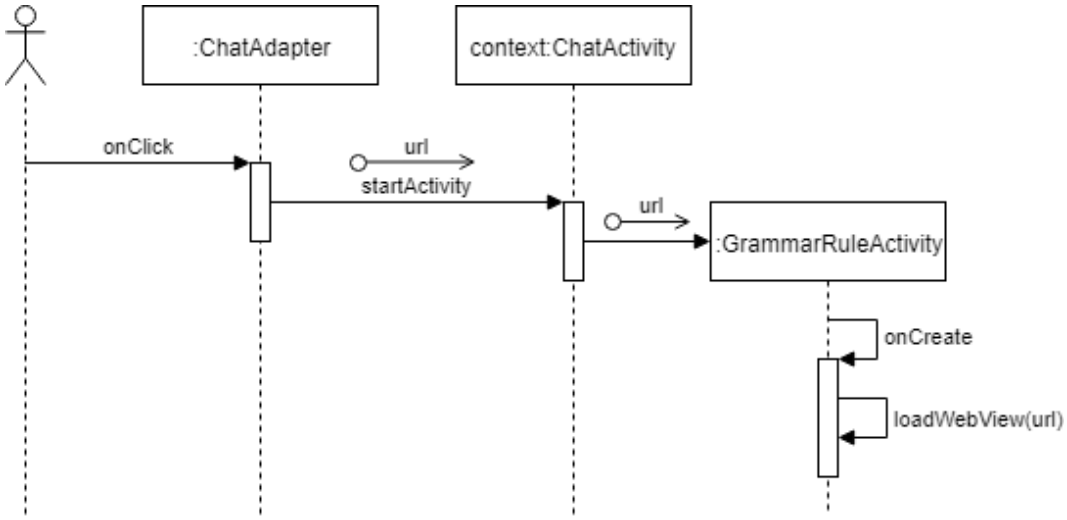
   Sequence Diagram Menampilkan Pesan Koreksi dapat dilihat pada gambar 3.34



Gambar 3.34 Diagram *Sequence* Menampilkan Pesan Koreksi

6. *Sequence Diagram* Menampilkan Aturan *Grammar*

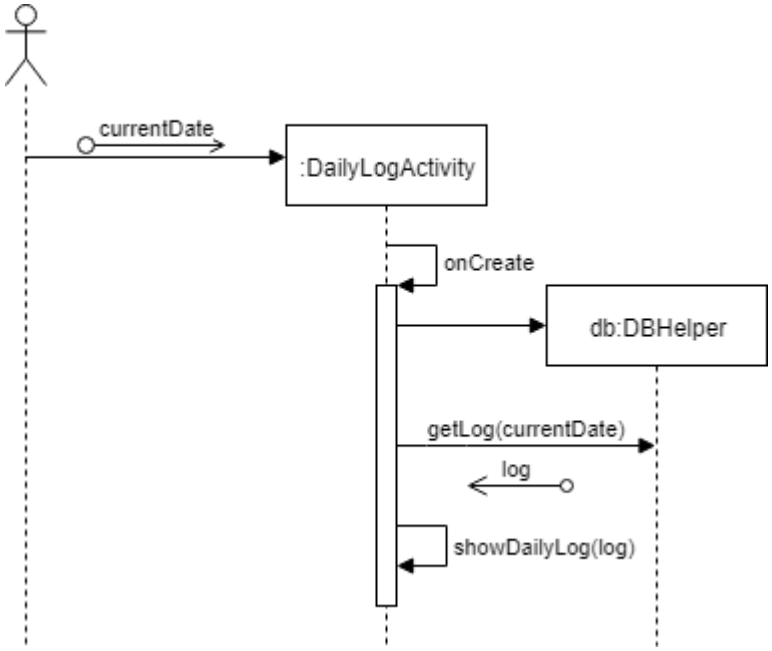
   Diagram Menampilkan Aturan *Grammar* dapat dilihat pada gambar 3.35



Gambar 3.35 Diagram *Sequence* Menampikan Aturan *Grammar*

7. *Sequence Diagram* Menampilkan Log Harian

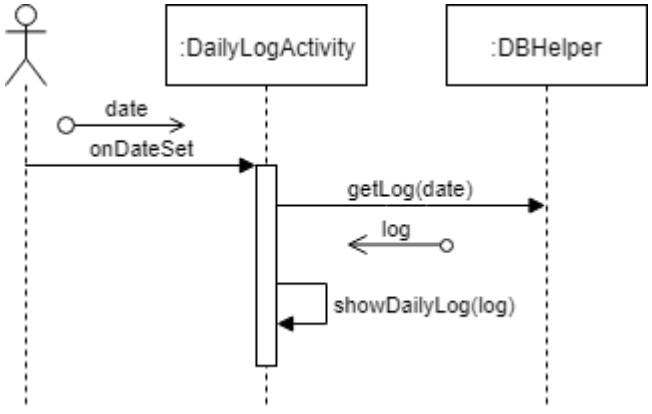
   Sequence Diagram Menampilkan Log Harian dapat dilihat pada gambar 3.36.



Gambar 3.36 Diagram *Sequence* Menampilkan Log Harian

8. *Sequence Diagram* Menampilkan Log Berdasarkan Tanggal

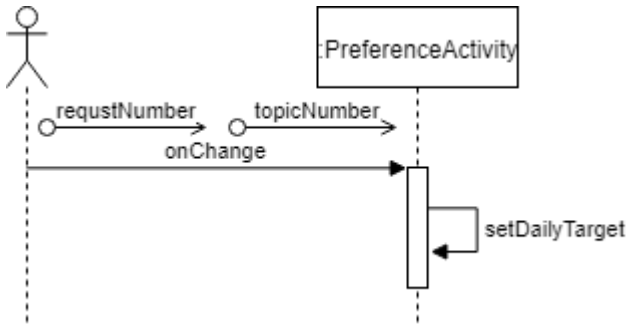
   Sequence Diagram Menampilkan Log Berdasarkan Tanggal dapat dilihat pada gambar 3.37.



Gambar 3.37 Diagram *Sequence* Menampilkan Log Berdasarkan Tanggal

9. *Sequence Diagram* Mengatur Target Harian

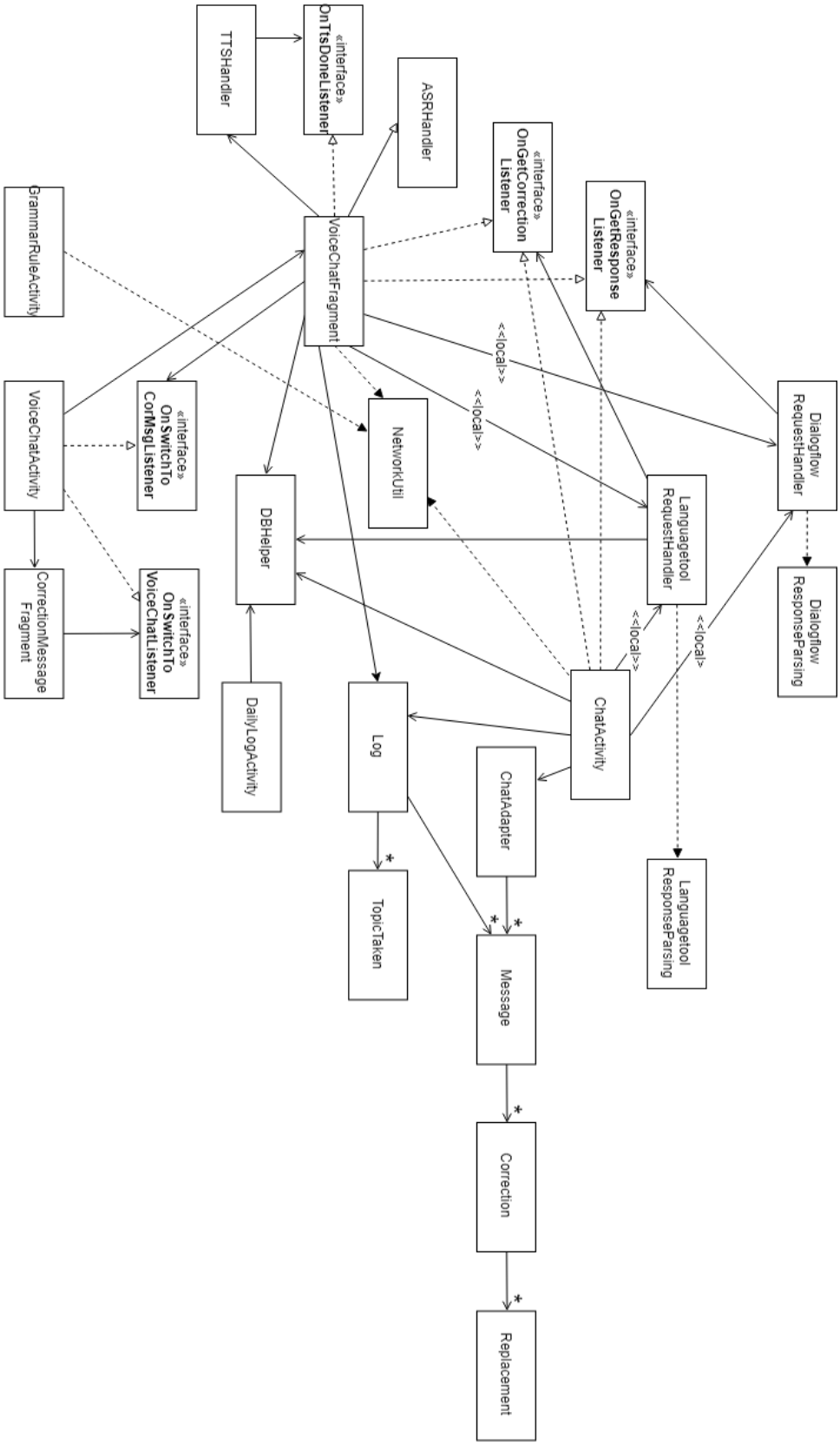
   Sequence Diagram Mengatur Target Harian dapat dilihat pada gambar 3.38.



Gambar 3.38 Diagram *Sequence* Mengatur Target Harian

3.3.7. Class Diagram

Dalam perancangan diagram kelas untuk keperluan analisis perangkat lunak, maka akan digambarkan kebutuhan yang akan diambil berdasarkan domain permasalahan dan tujuan dari pembuatan perangkat lunak. Perancangan diagram kelas secara keseluruhan dapat dilihat pada gambar 3.39.



Gambar 3.39 ***Class Diagram***

Berikut adalah informasi lebih detail untuk tiap-tiap kelas:

Tabel 3.15 Detail Kelas Interface OnGetCorrectionListener

<<interface>> **OnGetCorrectionListener**
– onGetCorrection

Tabel 3.16 Detail Kelas Interface OnGetResponseListener

<<interface>> **OnGetResponseListener**
– onGetResponse

Tabel 3.17 Detail Kelas Interface OnTtsDoneListener

<<interface>> **OnTtsDoneListener**
– onTtsDone

Tabel 3.18 Detail Kelas Interface OnSwitchToCorMsgListener

<<interface>> **OnSwitchToCorMsgListener**
– onSwitchToCorMsg – onSetCorMsgList

Tabel 3.19 Detail Kelas Interface OnSwitchToVoiceChatListener

<<interface>> **OnSwitchToVoiceChatListener**
– onSwitchToVoiceChat

Tabel 3.20 Detail kelas ChatActivity

ChatActivity
– dbHelper: DBHelper – chatAdapter: ChatAdapter – log: Log
onCreate() – sendMessage() – requestCorrection() + onGetCorrection() + onGetResponse() etc.

Tabel 3.21 Detail Kelas VoiceChatActivity

VoiceChatActivity
– vcFragment: VoiceChatFragment – corMsgFragment: CorrectionMessageFragment
+ onSwitchToCorMsg() + onSwitchToVoiceChat() etc.

Tabel 3.22 Detail Kelas VoiceChatFragment

VoiceChatFragment
– dbHelper: DBHelper – log: Log – ttsHandler: TTSHandler – listener: OnSwitchToVoiceChatListener
onCreateView() – listen() – processAsrResutl() – sendMessage() – requestCorrection() + onGetCorrection() + onGetResponse() etc.

Tabel 3.23 Detail Kelas CorrectionMessageFragment

CorrectionMessageFragment
– corMsgList: ArrayList<String> – corMsgAdapter: CorrectionMessageAdapter – listener: OnSwitchToCorMsgListener
+ onCreateView() + setCorMsgList() etc.

Tabel 3.24 Detail Kelas GrammarRuleActivity

GrammarRuleActivity
– webView: WebView
– loadWebView() etc.

Tabel 3.25 Detail Kelas DailyLogActivity

DailyLogActivity
– dbHelper: DBHelper
– showDailyLog() etc.

Tabel 3.26 Detail Kelas ASRHandler

ASRHandler
– myASR
+ listen() + processAsrResults() etc.

Tabel 3.27 Detail Kelas TTSHandler

TTSHandler
– myTTS
+ speak() + onInitListener() etc

Tabel 3.28 Detail Kelas LangagetooRequestHandler

LanguagetooRequextHandler
– dbHelper: DBHelper – msgId: int – userInput: String – listener: OnGetCorrectionListener
doInBackground() # onPostExecute() etc.

Tabel 3.29 Detail Kelas DialogflowRequestHandler

DialogflowRequestHandler
+ Listener: OnGetResponseListener
# doInBackground() # onPostExecute() etc

Tabel 3.30 Detail Kelas LanguagetooResponseParser

LanguagetoolResponseParser
+ getSimpleResponse()

Tabel 3.31 Detail Kelas DialogflowResponseParser

DialogflowResponseParser
+ getSimpleResponse()

Tabel 3.32 Detail Kelas DBHelper

DBHelper

+ onCreate() + getLog() + getTopic() + getAllMessage() + getCorrections() + getReplacements() + addNewLog() + addNewTopic() + addNewMessage() + addNewCorrection() + addnewReplacement() etc.

Tabel 3.33 Detail Kelas ChatAdapter

ChatAdapter
messages: Message[]
+ onCreateViewHolder() + onBindViewHolder() + getCount() + swapCursor() etc.

Tabel 3.34 Detail Kelas NetworkUtil

NetworkUtil
+ builtUrl() + getResponseFromHttpUrl()

Tabel 3.35 Detail Kelas Log

Log
– topicTakens: Topic[] – messages: Message[] – id: int – date:String – requestNumber:int – grammarFail:int – topicTaken:int etc.
+ getId() + setId() + getRequestNumber() + setRequestNumber() + getGrammarFail() + setGrammarFail() etc.

Tabel 3.36 Detail Kelas TopicTaken

TopicTaken
– Id: int – topicName: String – interactMode: int
+ getId() + setId() + getTopicName() + setTopicName() etc.

Tabel 3.37 Detail Kelas Message

Message
– corrections: Correction[] – id: int – logId: int – body: String – msgType: int – correctionOf: int – url: String
+ getId() + setId() + getLogId() + setLogId() + getBody() + setBody() etc.

Tabel 3.38 Detail Kelas Correction

Correction
– replacements: Replacement[] – id: int – messageId: int – message: String – offset: int – length: int – ruleId: int – subRuleId: int etc.
+ getId() + setId() + getOffset() + setOffset() + getLength() + setLength() etc.

Tabel 3.39 Detail Kelas Replacement

Replacement
– id: int – value: String
+ getId() + setId() + getValue() + setValue()

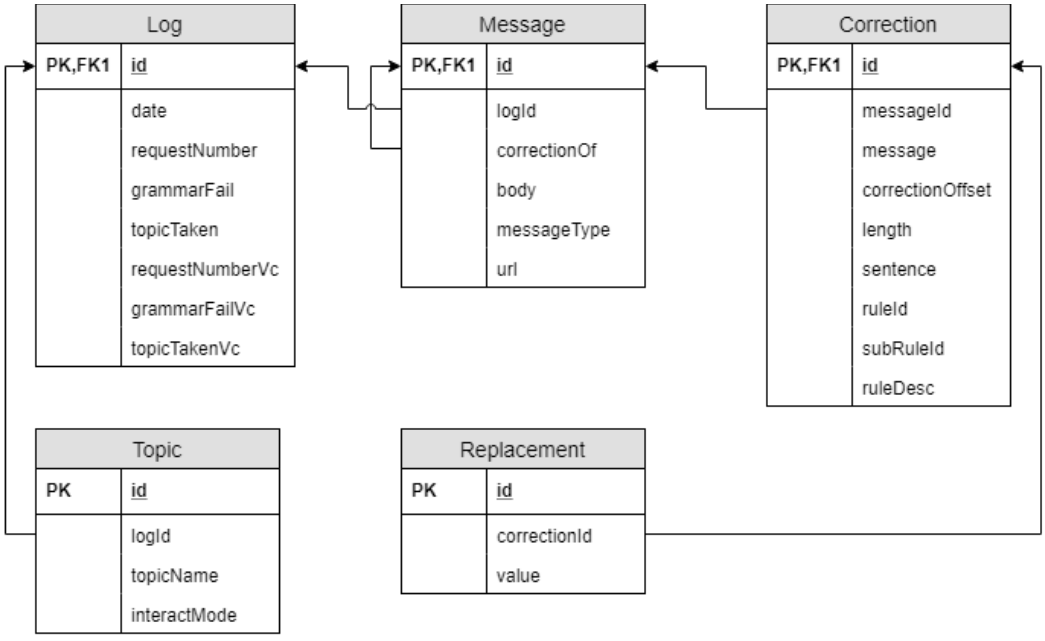
3.4. Perancangan Basis Data

Perancangan basis data yaitu menciptakan atau merancang data yang terhubung dan disimpan secara bersama-sama. Untuk menggambarkannya digunakanlah diagram relasi dan dan struktur tabel. Dari dua hasil tersebut, implementasikan basis data akan bisa dikerjakan.

3.4.1. Skema Relasi

Model data relasional merupakan model data dimana hubungan antar data, arti data dan batasannya dijelaskan dengan baris dan kolom. Secara formal, ke semuanya itu digambarkan ke dalam diagram relasi dan diagram skema.

Adapun untuk lebihnya tentang diagram relasi pada sistem *chatbot* pembelajaran bahasa inggris dapat dilihat pada gambar 3.40 sebagai berikut



Gambar 3.40 Skema Relasi

3.4.2. Struktur Tabel

Dibawah ini adalah struktur tabel yang digunakan dalam *chatbot* pembelajaran percakapan Bahasa Inggris yang terbagi ke dalam beberapa tabel, sebagai berikut :

3.4.2.1. Tabel Log

Tabel 3.40 Struktur Tabel Log

Nama *field*	***Type***	***Length***	**Keterangan**
id	INTEGER	-	PRIMARY KEY
date	DATETIME	-	
requestNumber	INTEGER	-	
grammarFail	INTEGER	-	
topicTaken	INTEGER	-	
requestNumberVc	INTEGER	-	
grammarFailVc	INTEGER	-	
topicTakenVc	INTEGER	-	

3.4.2.2. Tabel Message

Tabel 3.41 Struktur Tabel Message

Nama *field*	***Type***	***Length***	**Keterangan**
id	INTEGER	-	PRIMARY KEY
logId	INTEGER	-	FOREIGN KEY
correctionOf	INTEGER	-	
body	TEXT	-	
msgType	INTEGER	-	
url	TEXT	-	

3.4.2.3. Tabel Correction

Tabel 3.42 Struktur Tabel Correction

Nama *field*	***Type***	***Length***	**Keterangan**
id	INTEGER	-	PRIMARY KEY
messageId	INTEGER	-	FOREIGN KEY
message	TEXT	-	
correctionOffset	INTEGER	-	
length	INTEGER	-	
sentence	TEXT	-	
ruleId	TEXT	-	
subRuleId	INTEGER	-	
ruleDesc	TEXT	-	

3.4.2.4. Tabel Replacement

Tabel 3.43 Struktur Tabel Replacement

Nama *field*	***Type***	***Length***	**Keterangan**
id	INTEGER	-	PRIMARY KEY
correctionId	INTEGER	-	FOREIGN KEY
value	TEXT	-	

3.4.2.5. Tabel Topic

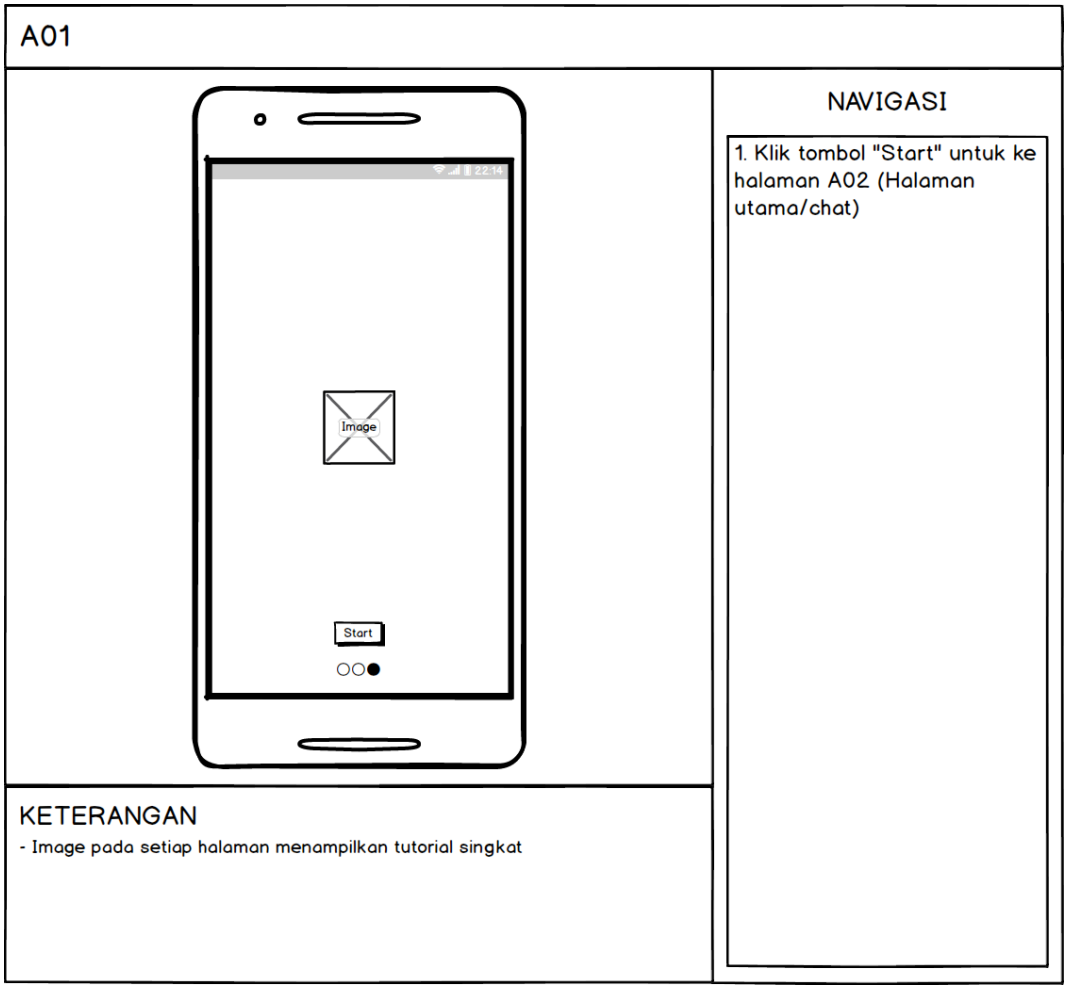
Tabel 3.44 Struktur Tabel Topic

Nama *field*	***Type***	***Length***	**Keterangan**
id	INTEGER	-	PRIMARY KEY
logId	INTEGER	-	FOREIGN KEY
topicName	TEXT	-	
interactMode	INTEGER	-	

3.5. Perancangan Arsitektur Sistem

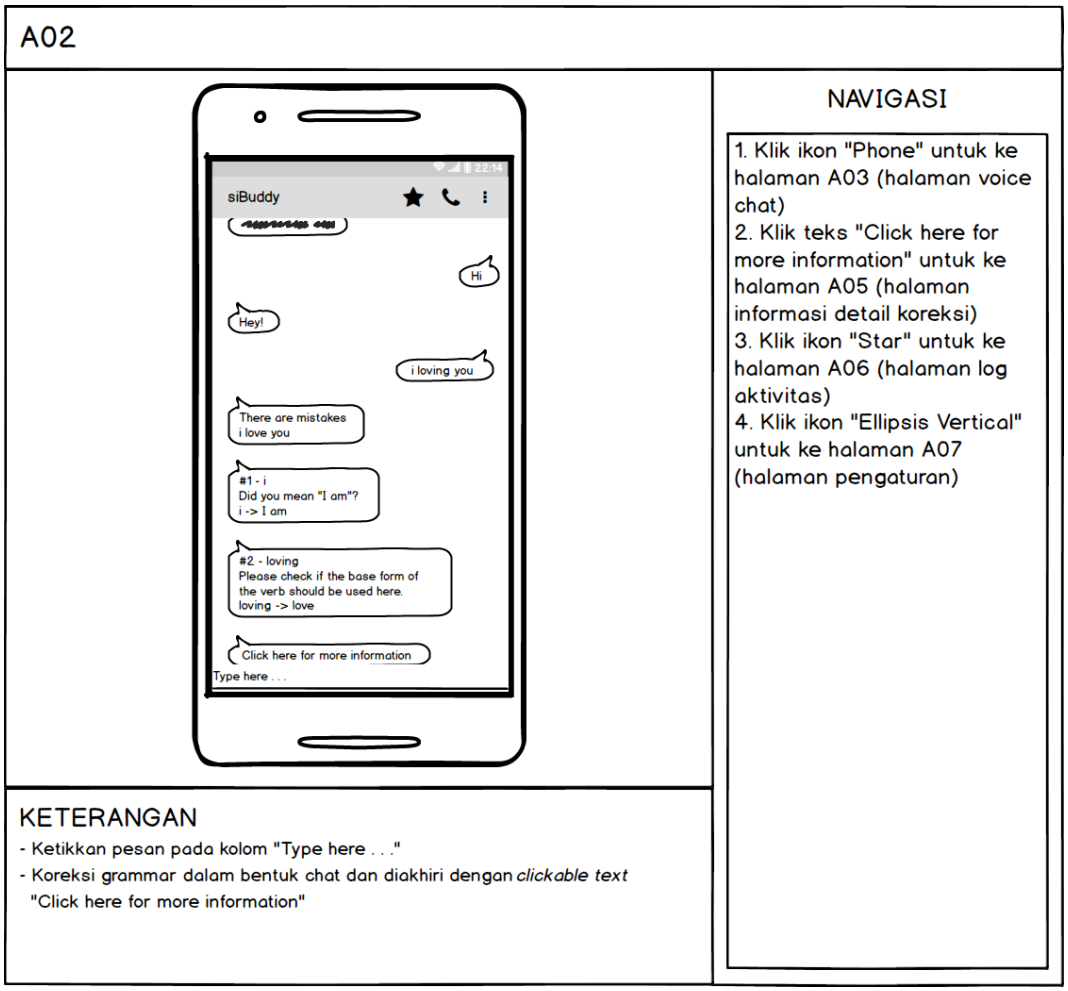
3.5.1. Perancangan Antarmuka

3.5.2.1. Perancangan antarmuka tampilan pembuka



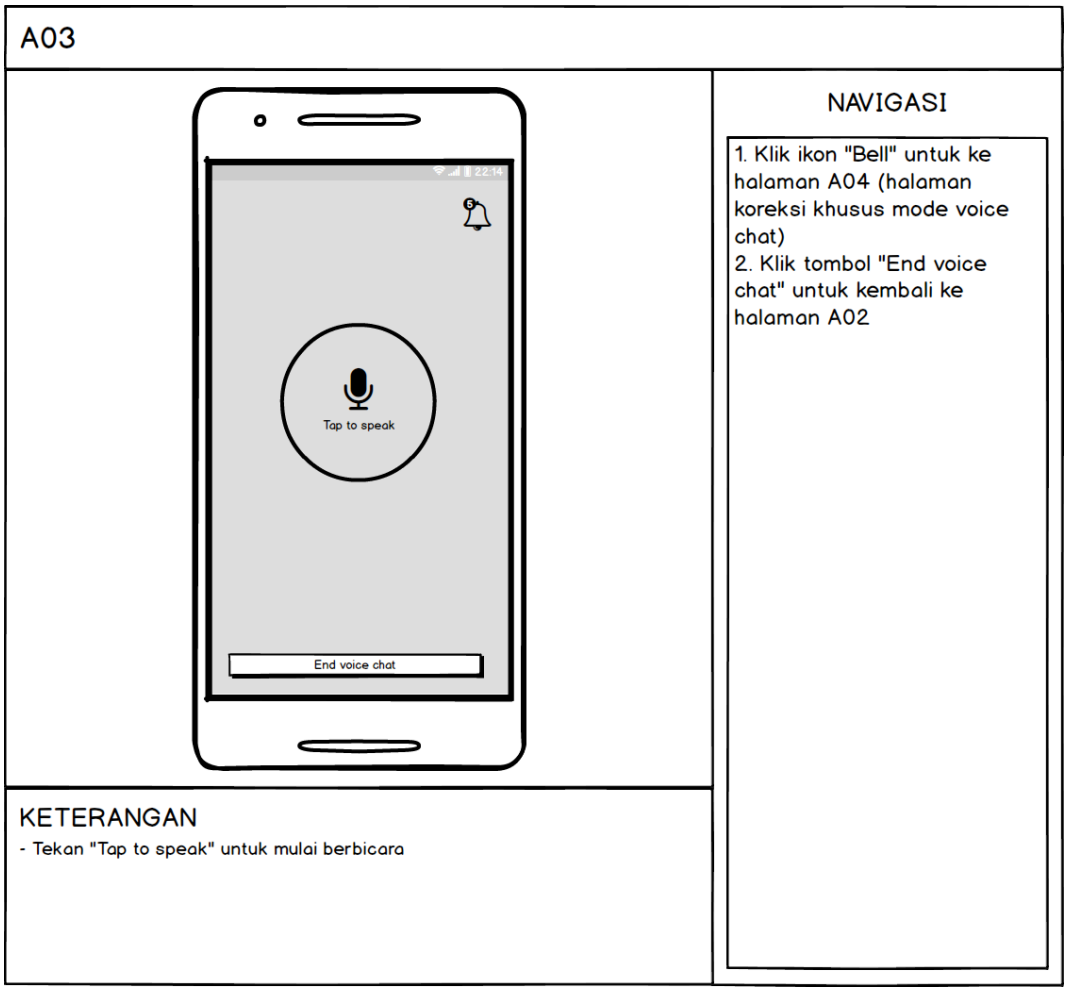
Gambar 3.41 Rancangan antarmuka tampilan pembuka

3.5.2.2. Perancangan antarmuka halaman *Chat*



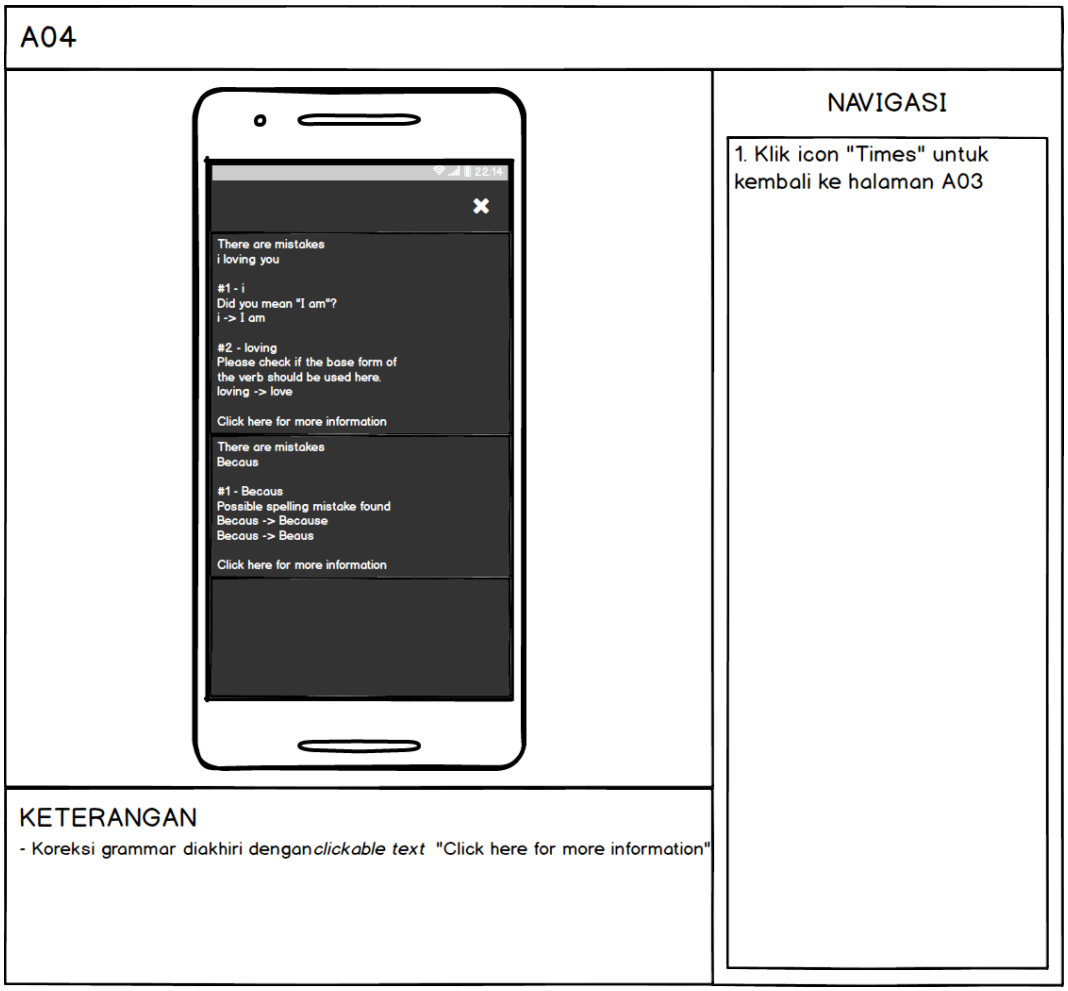
Gambar 3.42 Rancangan antarmuka halaman *Chat*

3.5.2.3. Perancangan antarmuka halaman *Voice chat*



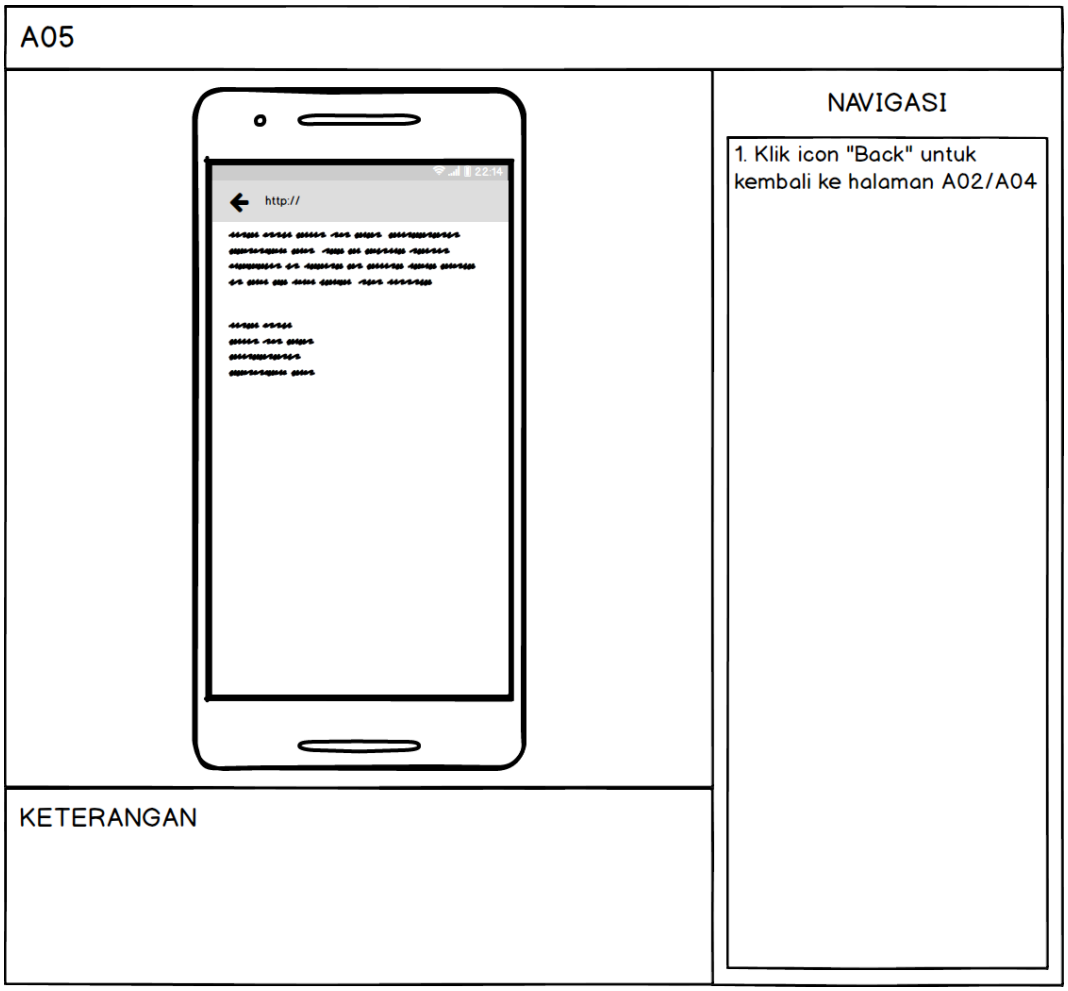
Gambar 3.43 Rancangan antarmuka halaman *Voice chat*

3.5.2.4. Perancangan antarmuka halaman *Correction message*



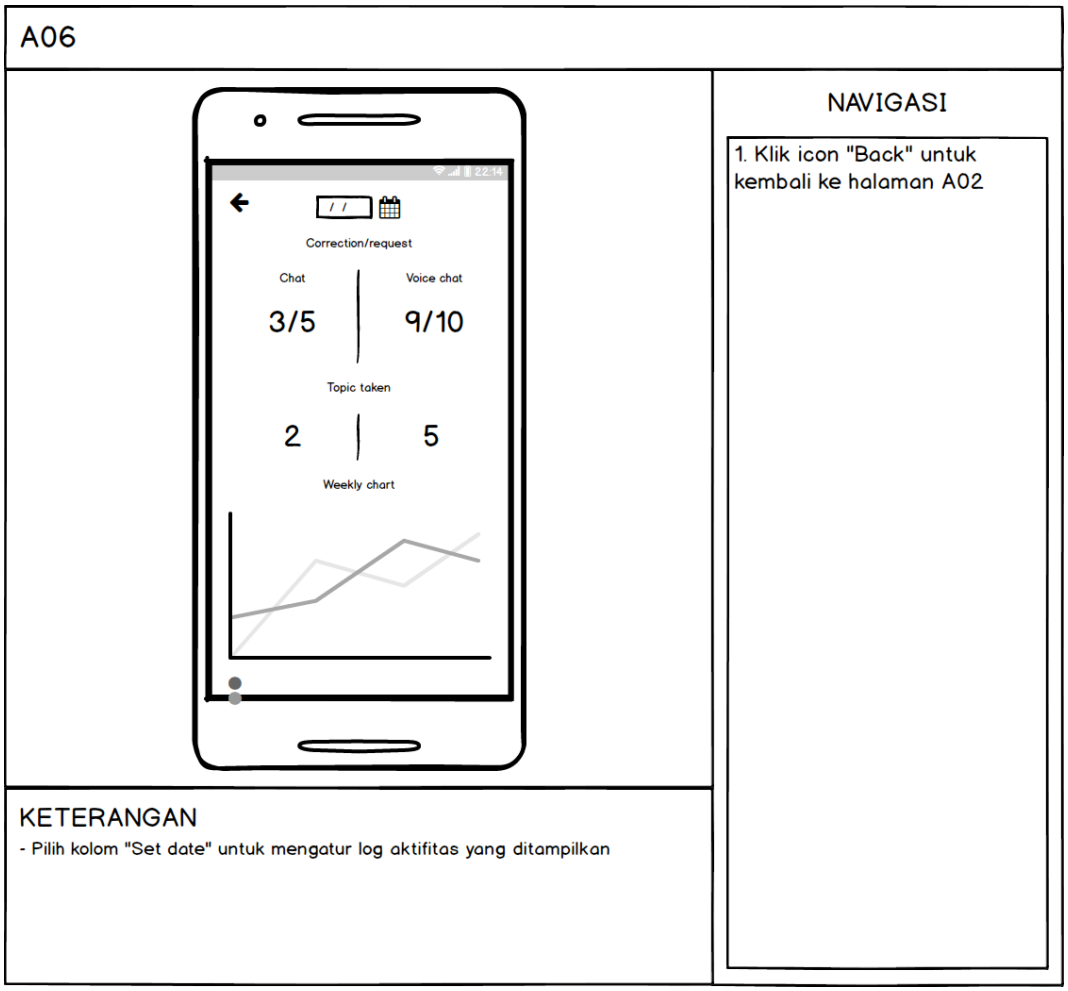
Gambar 3.44 Rancangan antarmuka halaman *Correction message*

3.5.2.5. Perancangan antarmuka halaman Aturan *Grammar*



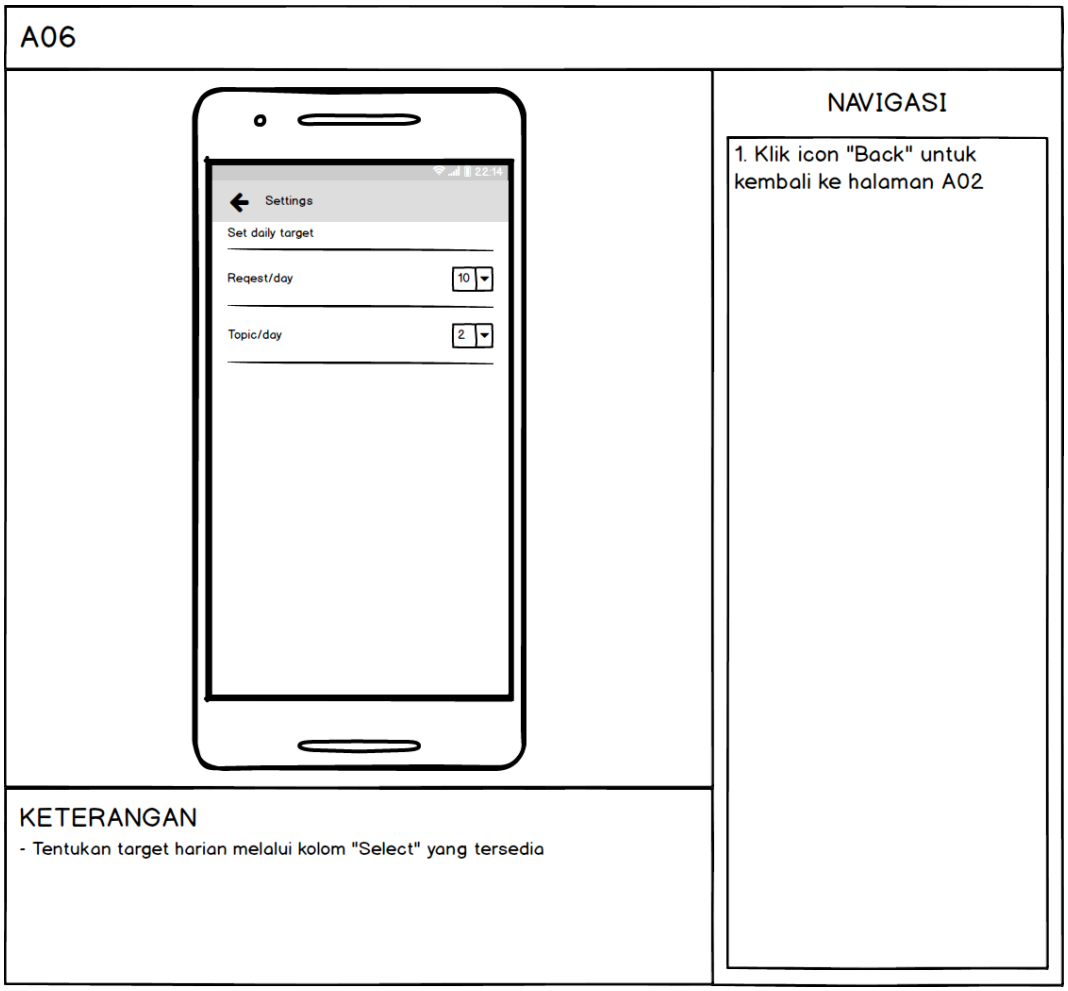
Gambar 3.45 Rancangan antarmuka halaman Aturan *Grammar*

3.5.2.6. Perancangan antarmuka halaman *Daily log*



Gambar 3.46 Rancangan antarmuka halaman *Daily log*

3.5.2.7. Perancangan antarmuka halaman *Set Daily Target*



Gambar 3.47 Rancangan antarmuka halaman *Set Daily Target*