Final Year Project Final Report

[FYP19059] Automated Question Answering with Chatbots

COMP4801 Final Year Project

Department of Computer Science

University of Hong Kong

LEE N*** ***

Team Member	UID
SUNG ***	***
KONG ***	***

Supervisor

Dr. TAM Anthony T.C.

2 May 2020

Abstract

With great improvement on natural language processing, the chatting bots are replacing human labor in enquiry. The University of Hong Kong Libraries wants its own chatbot responding to user enquiry implemented on their website to assist their workers and expand its user accessibilities. The scope of the information is discussed with the clients to maximize the user utilization. This project builds the flowchart, the question understanding model with Dialogflow and the database constructed with information provided on the libraries website, deploys the question-respond matching functions with Dialogflow webhook and Firebase and implements the ultimate product on the libraries website. The designs of the NLU model and the database are optimized to the specific purpose. Accurate classification of the user input questions can be achieved by NLP on Dialogflow. This report presents the procedure that had been done during the two academic semesters and evaluates the result of the product.

Table of Contents

List of Figures

List of Abbreviations

Abbreviation	**Definition**
AI	Artificial Intelligence
API	Application Program Interface
HKUL	The University of Hong Kong Libraries
ML	Machine Learning
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
UI	User Interface
UX	User Experience

I. INTRODUCTION

This document is to report the detailed process of the project “Automated Question Answering with Chatbots” that have been done in two semesters from 2019 September to 2020 April. The University of Hong Kong Libraries first suggested the project to create a chatting bot that answers some queries from HKUL users.

The University of Hong Kong Libraries started from two rooms in the Main Building of HKU in 1912 and has expanded to the Main Library and six branches with more than 3 million volumes of documents and 6 million digital documents today [1]. It does not only provide the informative resources but also the learning spaces such as study tables, rooms, and computers in *Level 3* and newly opened *Ingenium* and *Think Tank* in the Main Library, orientations, workshops and more [1]. Expanding services also induce more questions from users. HKUL currently relies on human resources to respond to inquiries; visit, email, phone call, WhatsApp or video call. In the first meeting, librarians introduced their previous attempts on social network services, such as Twitter, Facebook and Instagram, to interact with their users. However, HKUL accounts had low accessibilities to users, and the librarians suggested this project as an alternative. HKUL also had tried the chatbot project using the WhatsApp inquiry data, but it was not successful either.

I.1 Technical Background of Chatbot

Chatbot is an automated program build to talk to humans in normal human languages, and one application of natural language processing. Chatbot technology is rapidly improving as other NLP applications with machine learning. NLP is one of AI technology for linguistic interactions between humans and computers [2]. Traditional programming techniques were attempted to let computer programs and humans communicate with natural languages, but the performance was not satisfying since codifying natural languages is very complicated and time consuming with too many variations and exceptions.

On the other hand, machine learning lets the machines to find patterns from massive amount of data given and build models. In other words, programmers do not have to manually design the language understanding models but computers can do it. In addition, more input data may improve the model accuracies. Accordingly, the performance of NLP has been remarkably advanced by ML, such as that of Amazon's Alexa and Microsoft's Xiaoice.

Chatbot naturally has improved with NLP as well. Better understanding of human languages and producing more humanlike languages by machines is achieved by improvements of NLP. There are wide range of chatbots by purpose from answering few frequently asked questions on a website to having emotional communications. The diversity diverges the structure of the models, databases, and UIs heavily by each project.

I.2 Objectives and Scope of the Project

The chatbot which is easily accessible from HKUL website is expected to effectively filter user enquiries on the first stage; give answers or links to pages when the inquiries are related to information already shared on the website or give contact to talk to the librarians when the inquiries require further attentions. When the chatbot can effectively accomplish such function, it is able to successfully share workloads with the librarians.

The chatbot will cover some information available on the HKUL website. The website has broad range of information as the HKUL services expand, and some are less frequently asked than others. Therefore, the members of the project first suggested to focus on the following information;

1. Basic information of branches – This part comprises address, contact numbers and emails, opening hours, etc. of the Main Library and branch libraries.
2. User policies – Different registration policies are applied to different people, and document loan, request, and renew policies also differ by user types. It is not easy for users to find such information from the HKUL website because of the website design. Therefore, the chatbot will be designed to provide corresponding information to the users by the effectively designed database.
3. Facility policies – Diverse facilities serviced by HKUL are used by many of its users. However, the policies to use and book them is user-type dependent. The chatbot will let the users to get such information in simpler manner.
4. Document search – If chatbot can get data from the HKUL documentation API, it can function as document searching engine as well. HKUL already has a website for this function, Find@HKUL (find.lib.hku.hk), but the chatbot can support it by offering few of the key information, such as title, author, location and current availability. URL of Find@HKUL or contact to the librarians can be given at the end for further process.

The scope was shared with HKUL on November, and some modifications were made afterwards through discussions. Changes in scope will be explained in III.1 Flowchart.

The chatbot is expected to be the first communication channel to all the library users. Therefore, the interface should be straight-forward to everyone and the chatbot needs to have good accessibility. Therefore, pop up window on the right bottom corner of the official website was decided to be the most suitable integration. To keep the accessibility, the chatbot window should be visible on any pages on the HKUL website but can be minimized not to disturb the views. Such integration will allow the users to ask questions whenever they cannot find the information they need on the website.

I.3 Overview of This Report

This report shows the entire process taken in the project. Introduction section described the background of the project, current chatbot industry and the brief objectives and scope of the project. Next **METHODOLOGY** section will discuss the detailed procedure that was planned to be used and **RESULT** section will demonstrate any changes made from METHODOLOGY and the performance of the final product.

II. METHODOLOGY

This section introduces the detailed plans with selected tools to be used to accomplish the objectives of the project. The essential technology of chatting bots is NLP that allows the bots to understand the input questions and determine the processes required to output the desired response. As briefly explained in the **INTRODUCTION** session, NLP is ML based technology that can perform in higher accuracy by training with more data. Therefore, existing NLP tools are practical methods in chatbot projects to understand the user questions. Before deciding the tools, a google account was created and shared between the members. Because it is a cooperative project among three members especially during Hong Kong's protests and COVID-19, online accessibilities of the tools by multiple users were essential. All selected tools are connected to the Google account, and it will be shared to HKUL for further maintenance.

II.1 Tools Selection

First, utilizing chatbot platforms or frameworks were agreed since work from scratch would not guarantee the satisfying accuracy. Using the existing natural language understanding model was considered more efficient to accelerate the process with satisfactory final product. Among many existing chatbot frameworks, Dialogflow from Google was selected for NLU model development. Dialogflow is the web-based AI development framework for human-computer interactions supported by Google. It analyzes the patterns of input questions with Google's Natural Language API, which is one of the most developed in current state. Dialogflow is accessible by Google accounts and the model updates instantly, so simultaneous work by members would not cause confusions. Dialogflow originally uses Firebase for webhook inline editor and shows logs on Firebase. Firebase is the cloud-based development platform from Google Cloud. It allows to manage the database on Google Cloud. Inline editor would be used, and maintenance of the database on the cloud system was preferable in this project for consistency in project development and afterward maintenance by the libraries. Therefore, Firebase was chosen to store database and log history.

Before introducing the specific processes, shared concepts across all processes should be explained first. ‘Intents’ are the categories of input questions in Dialogflow. An intent is triggered when the input question from a user has more than 50% matching accuracy to the intent based on the trained phrases. ‘Entities’ are the parameters that should be extracted from the user questions. Many parameters can be set in each intent, and each may be set to be required or not. Fulfillment is the codes to be built to call data from database and form complex responses with or without such data.

To design the workflow of the entire project with selected tools, the chatbot flowchart needed to be created first. Thorough design of the database and the NLU model is crucial to build a successful chatbot since the database and the NLU model should synchronize. Small differences such as capitalized and uncapitalized letters can cause errors, so the reference showing how the different steps are related to each other is needed. In addition, the project is collaborative work between three members, but the design can be frequently changed. Therefore, the tool that is able to show the correlation between different steps of the process and be shared online was needed. Web mind mapping tool Mindomo satisfied all conditions and was chosen to be used to design the flowchart.

II.2 Flowchart

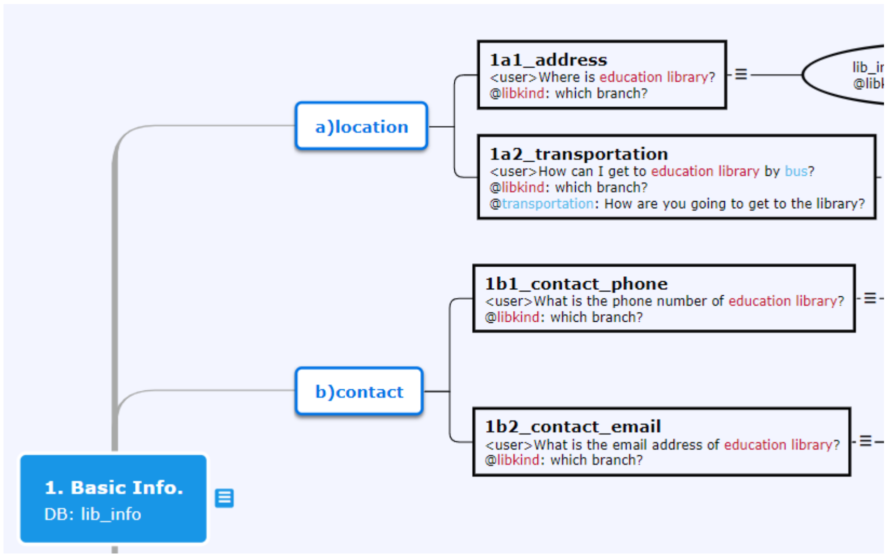

1. Basic Info.
DB: lib_info
a)location
b)contact
1a1_address
<user>Where is education library?
@libkind: which branch?
1a2_transportation
<user>How can I get to education library by bus?
@libkind: which branch?
@transportation: How are you going to get to the library?
1b1_contact_phone
<user>What is the phone number of education library?
@libkind: which branch?
1b2_contact_email
<user>What is the email address of education library?
@libkind: which branch?

Figure II.1 Intent Categorization

The flowchart was divided into first the 4 categories of scope, second the similar question topic, and third specific questions to be built as intents. Figure *II.1* shows the example of intent categorization; ‘Basic_info’ is the first category, ‘location’ and ‘contact’ are the second categories, and ‘address’, ‘transportation’, ‘contact_phone’ and ‘contact_email’ are the third categories, intents, of specific questions.

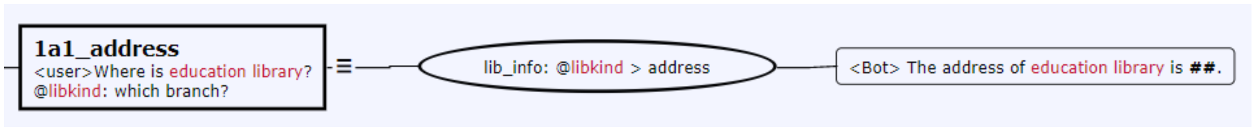

1a1_address
<user>Where is education library?
@libkind: which branch?
lib_info: @libkind > address
<Bot> The address of education library is ##.

Figure II.2 Flowchart Design

Each question needed to be trained separately as an intent. Figure II.2 shows the general flowchart of each intent. The bold boxes indicate intents in the flowchart. Intents were designed with intent name, sample question, and parameters needed. Intent names were numbered in the same way the intents were categorized since intents are alphabetically ordered in Dialogflow console. Prompt questions needed for the required entities were included next to parameters. The ellipse shows the database searching process, by the order of keys. Intents that did not require database connection does not have ellipse in the flowchart. The example of Figure II.2 shows that it needs to go to ‘lib_info’ database and find corresponding branch name from ‘libkind’ entity as the first key, then ‘address’ as the second key. The third round edged box shows the sample response. ## shows the data got from previous ellipse. Each entity was assigned to a unique font color, and corresponding information from the sample questions and the response has same colors to easily notice same entities and modify the entities or fulfillment if needed. For example, ‘libkind’ entity is in red, and one of the values of the entity ‘education library’ is also in red in Figure *II.2*. The values may be reused in the response as the example.

Registration intent ‘card_get’ needed many follow up questions to provide accurate response to the users, which required complicated workflow. The flowchart of intents of card registration is shown in Figure *II.3*. The intent followed by ‘card_get’ intent depends on the user types. When the user question did not include any parameter values of ‘card_get’ intent, ‘card_get’ intent should ask for such information, and one of the intents followed right after ‘card_get’ would be triggered depending on the user response. If any one of the entities was included already, the flow should follow the blue arrows in Figure *II.3*. Different workflow was needed for different users since the types of available library card types depend on user types. For the users who can apply for multiple types of cards, like graduates, the bot should ask the card they want and form the response regarding the card type. On the other hand, some users can only register to one type of card like part time staffs. For those users, the bot should answer as soon as it fetches the user type without asking the card types the user want.

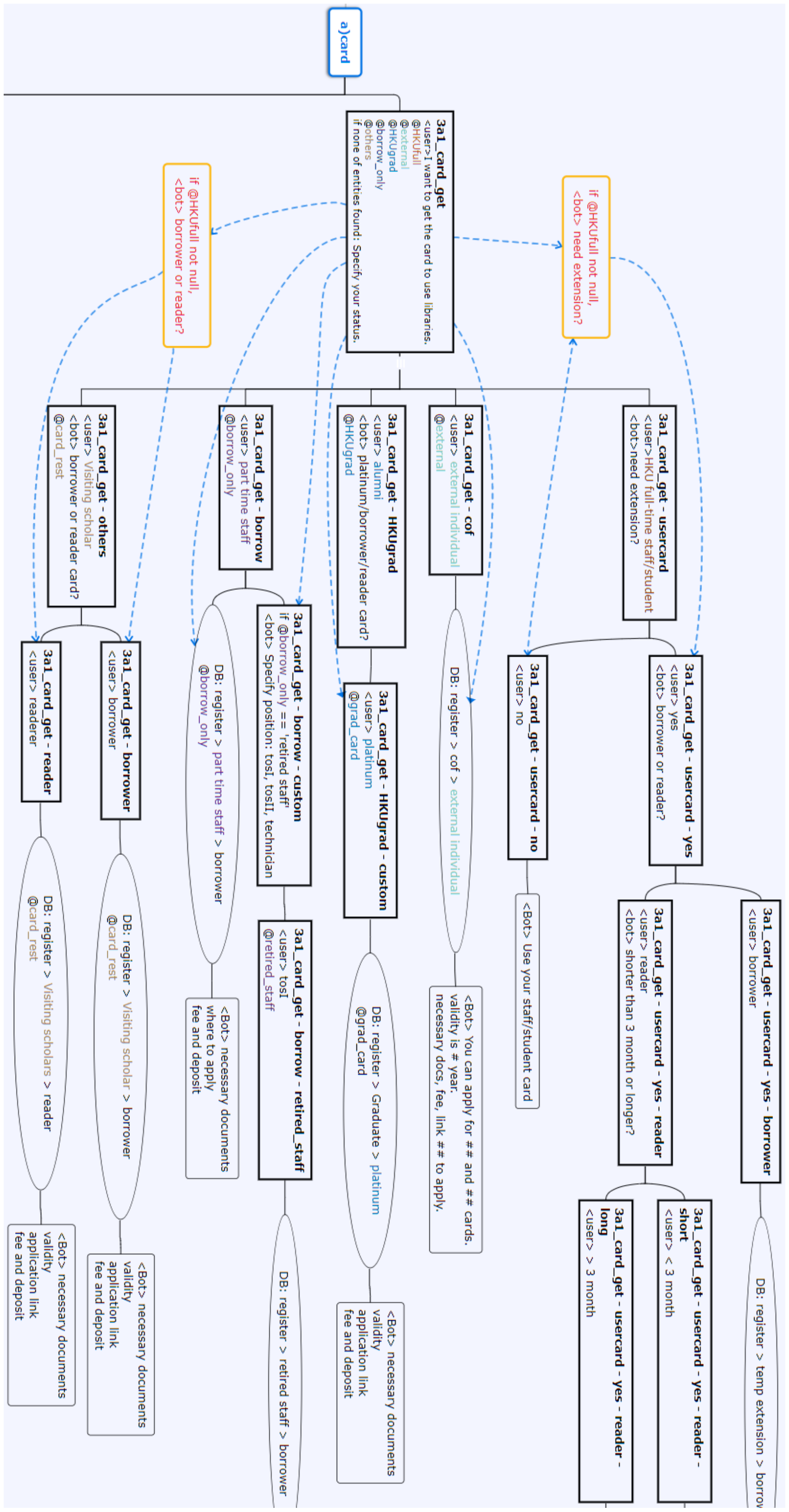
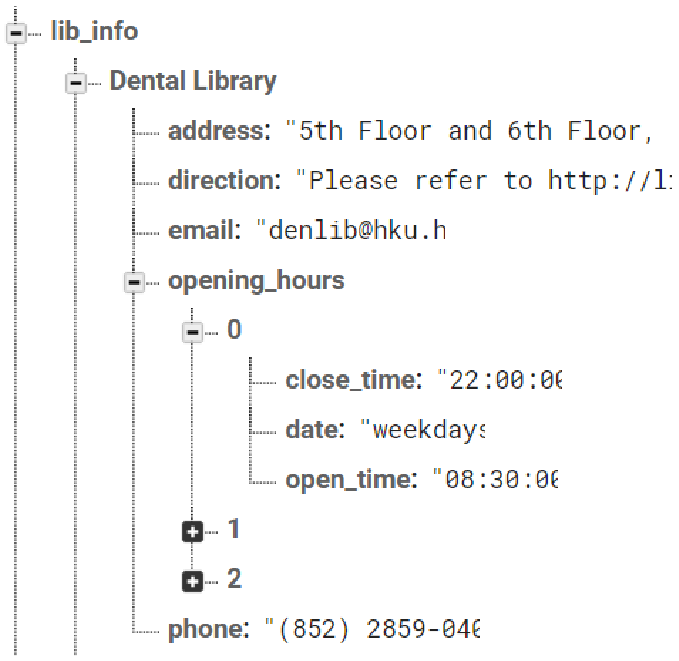

a)card
3a1_card_get
<user> I want to get the card to use libraries.
@HKUfull
@external
@HKUgrad
@borrow_only
@others
if none of entities found: Specify your status.
if @HKUfull not null,
<bot> borrower or reader?
if @HKUfull not null,
<bot> need extension?
3a1_card_get - usercard
<user>HKU full-time staff/student
<bot>need extension?
3a1_card_get - cof
<user> external individual
@external
3a1_card_get - HKUgrad
<user> alumni
<bot> platinum/borrower/reader card?
@HKUgrad
3a1_card_get - borrow
<user> part time staff
@borrow_only
3a1_card_get - others
<user> Visiting scholar
<bot> borrower or reader card?
@card_rest
3a1_card_get - usercard - yes
<user> yes
<bot> borrower or reader?
3a1_card_get - usercard - no
<user> no
DB: register > cof > external individual
3a1_card_get - HKUgrad - custom
<user> platinum
@grad_card
3a1_card_get - borrow - custom
if @borrow_only == 'retired staff'
<bot> Specify position: tosI, tosII, technician
DB: register > part time staff > borrower
@borrow_only
3a1_card_get - borrower
<user> borrower
3a1_card_get - reader
<user> readerer
3a1_card_get - usercard - yes - borrower
<user> borrower
3a1_card_get - usercard - yes - reader
<user> reader
<bot> shorter than 3 month or longer?
<Bot> Use your staff/student card
<Bot> You can apply for ## and ## cards.
validity is # year.
necessary docs, fee, link ## to apply.
DB: register > Graduate > platinum
@grad_card
3a1_card_get - borrow - retired_staff
<user> tosI
@retired_staff
<Bot> necessary documents
where to apply
fee and deposit
DB: register > Visiting scholar > borrower
@card_rest
DB: register > Visiting scholars > reader
@card_rest
DB: register > temp extension > borrow
3a1_card_get - usercard - yes - reader - short
<user> < 3 month
3a1_card_get - usercard - yes - reader - long
<user> > 3 month
<Bot> necessary documents
validity
application link
fee and deposit
DB: register > retired staff > borrower
<Bot> necessary documents
validity
application link
fee and deposit
<Bot> necessary documents
validity
application link
fee and deposit

Figure II.3 flowchart of card registration intents

II.3 Database

Firebase is a NoSQL fully managed database and services two types of database, Realtime database and Cloud Firestore. Realtime database is one big JSON tree while Cloud Firestore is a document database supporting auto-scaling. The database was designed to contain data extracted and organized by the members from static information on HKUL website, and keys such as entity values were used to collect data from the database. Therefore, JSON structure with keys and values was more suitable for this project, and the database was constructed on the Firebase Realtime database. Firebase web console supports adding data but not editing the keys. Therefore, database construction was done by exporting the JSON file to get the updated database, edit again and import back to Firebase.

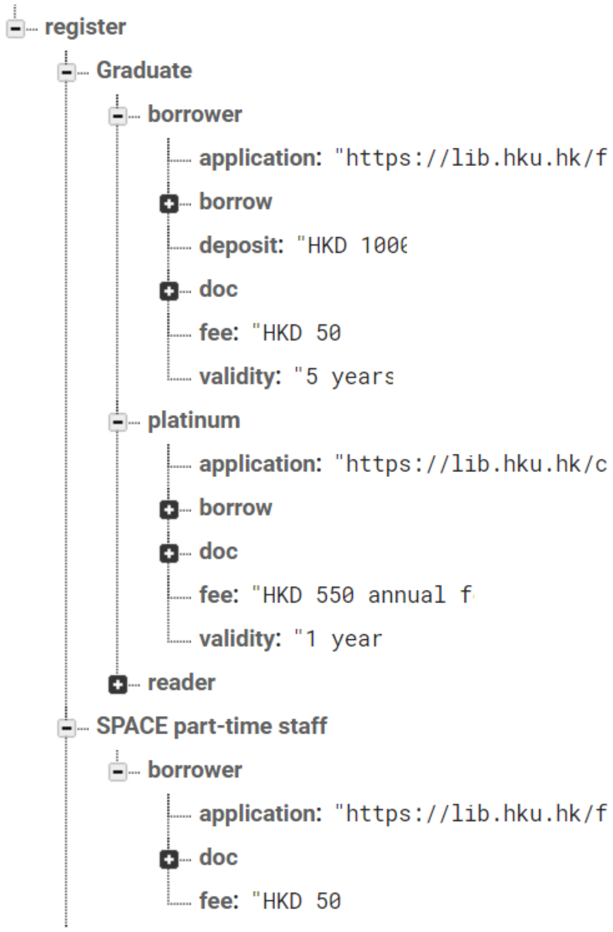


```
lib_info
    Dental Library
        address: "5th Floor and 6th Floor,
        direction: "Please refer to http://l
        email: "denlib@hku.h
        opening_hours
            0
                close_time: "22:00:0
                date: "weekday
                open_time: "08:30:0
            1
            2
        phone: "(852) 2859-04
```

Figure II.4 Database Structure of ‘lib_info’

Database structure also matched to the flowchart. The order of keys was as defined in the ellipses, and the structure afterward depended on the efficiency for searching. Figure II.3 shows the database structure of first scope, basic library information. It followed the basic structure defined in Figure *II.2*, ‘lib_info’, branch, then ‘address’. ‘opening_hours’ is an example of data that needs further structure. Opening hours of the libraries varies by days of the week, weekdays, Saturdays and Sundays. Therefore, they were separated by keys 0 – weekdays, 1 – Saturdays or 2 – Sundays.

On the other hand, some database such as ‘register’ needed more complicated design compared to ‘lib_kind’ due to the nature of the information. Registration information to different users were organized into ‘register’ database. Registration information and available card types were user-type dependent. Card types did not have equivalent information either. Many information was missing in some users, so the most complicated one was chosen as the criterion first. Then, NULL values were entered to missing information to keep the constant structure. Some user types were allowed to make only one type of card, so card type was excluded from the database first. However, for consistency, the structure of register database is register/‘user_type’/‘card_type’/ ‘information_title’.



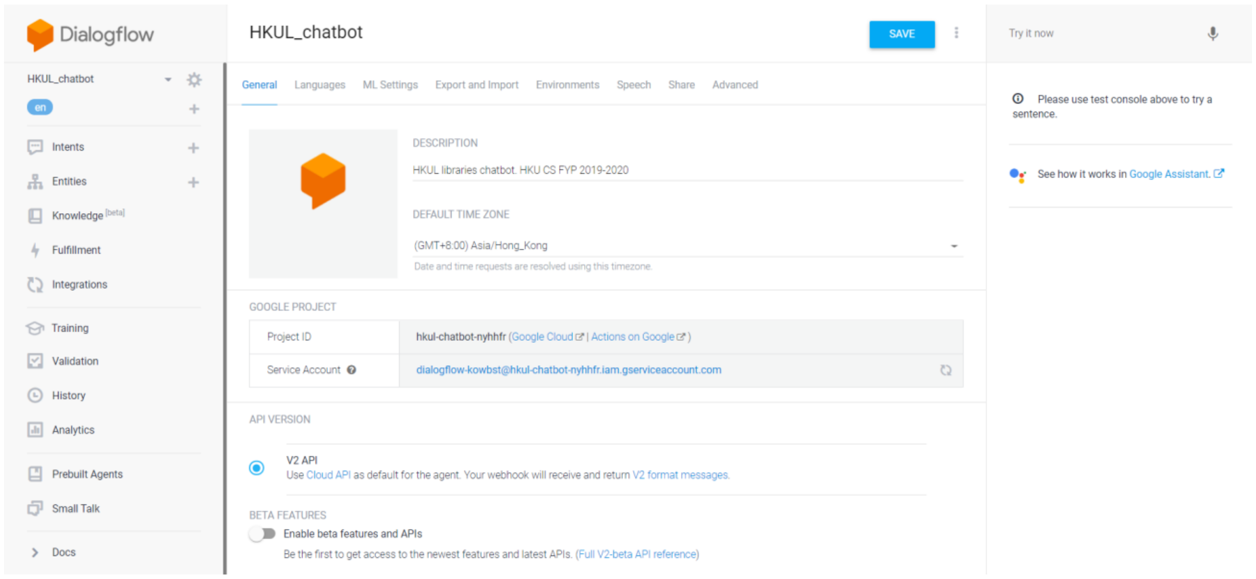
```
register
    Graduate
        borrower
            application: "https://lib.hku.hk/f
            borrow
            deposit: "HKD 100
            doc
            fee: "HKD 50
            validity: "5 years
        platinum
            application: "https://lib.hku.hk/c
            borrow
            doc
            fee: "HKD 550 annual f
            validity: "1 year
        reader
    SPACE part-time staff
        borrower
            application: "https://lib.hku.hk/f
            doc
            fee: "HKD 50
```

Figure II.5 Database Structure of ‘register’

From Figure *II.5*, two user types ‘Graduate’ and ‘SPACE part-time staff’ have different number of available card types; the prior has ‘platinum’, ‘borrower’ and ‘reader’ card while the latter only has ‘borrower’ card. Also, the numbers of information on each card type differ. Graduate borrower card has 6 kinds of information, while platinum card does not include ‘deposit’, and the borrower card of SPACE part-time staff only has three kinds of information. With this structure, for instance, when ‘validity’ of SPACE part-time staff’s borrower card is searched, NULL result will be returned. Therefore, database searching algorithm could have conditions to NULL values, and the response could be formed according to conditions.

II.4 Dialogflow

All the key functions of a chatting bot had been implemented by Dialogflow. Figure *II.6* shows the Dialogflow console. The list on the left shows the tabs of intents, entities, fulfillment, etc. The middle part provides the working area for the selected tabs. There is a testing console on the right which was used in testing and debugging.



DESCRIPTION

HKUL libraries chatbot. HKU CS FYP 2019-2020

DEFAULT TIME ZONE

(GMT+8:00) Asia/Hong_Kong

Date and time requests are resolved using this timezone.

GOOGLE PROJECT

Project ID	hkul-chatbot-nyhhfr (Google Cloud | Actions on Google)
Service Account	dialogflow-kowbst@hkul-chatbot-nyhhfr.iam.gserviceaccount.com

API VERSION

● V2 API
Use Cloud API as default for the agent. Your webhook will receive and return V2 format messages.

BETA FEATURES

Enable beta features and APIs
Be the first to get access to the newest features and latest APIs. (Full V2-beta API reference)

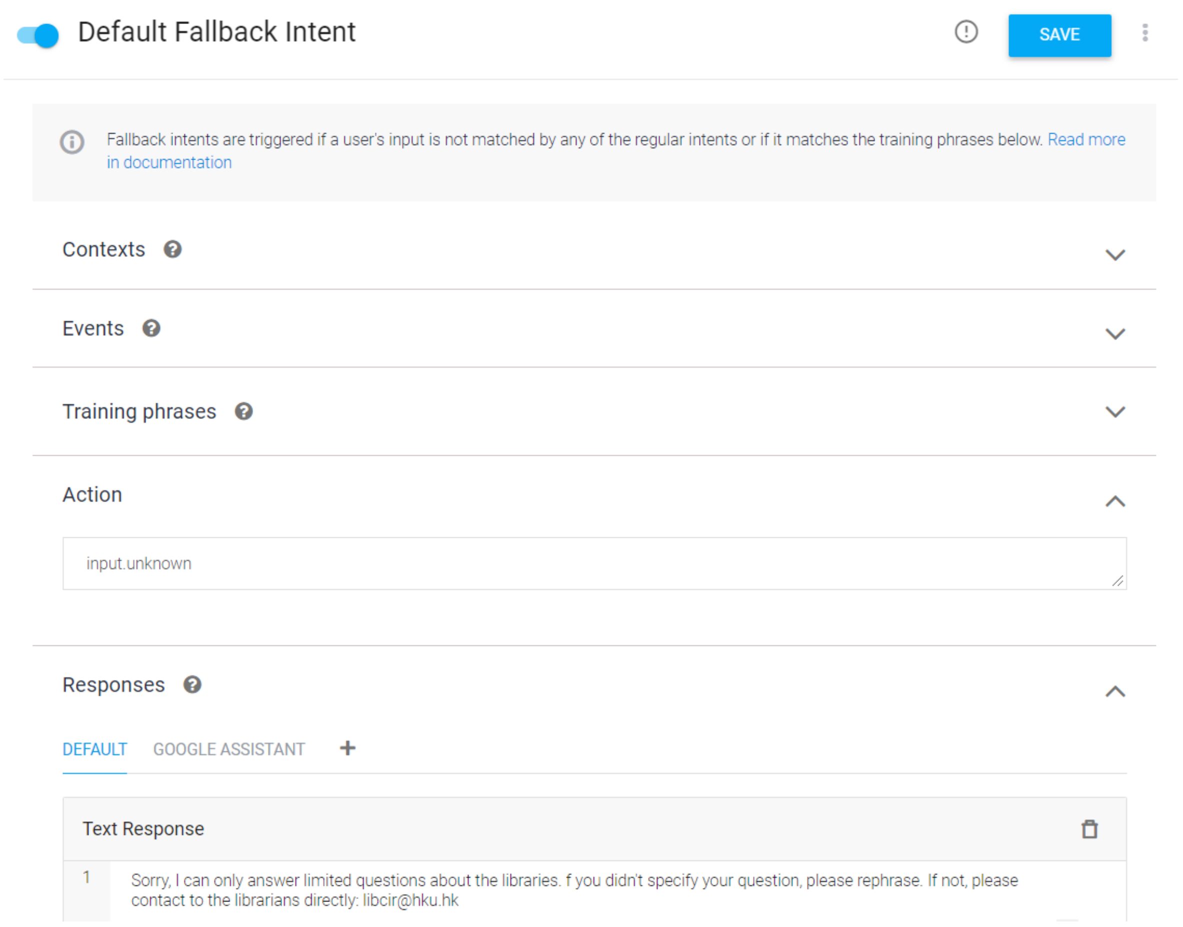
Please use test console above to try a sentence.

See how it works in Google Assistant.

Figure II.6 Dialogflow console

NLU model building started by constructing the NLU model with entities and intents. Then, static responses were added in the required intents. For intents requiring dynamic responses, webhook was enabled first then the code on fulfillment was written. Dialogflow testing console was used to test the performances of the NLU model and deployed algorithms on fulfillment. Following sections describe the detailed methodology of each process.

II.4-1 NLU Model


Default Fallback Intent

SAVE

Fallback intents are triggered if a user's input is not matched by any of the regular intents or if it matches the training phrases below. Read more in documentation

Contexts

Events

Training phrases

Action

input.unknown

Responses

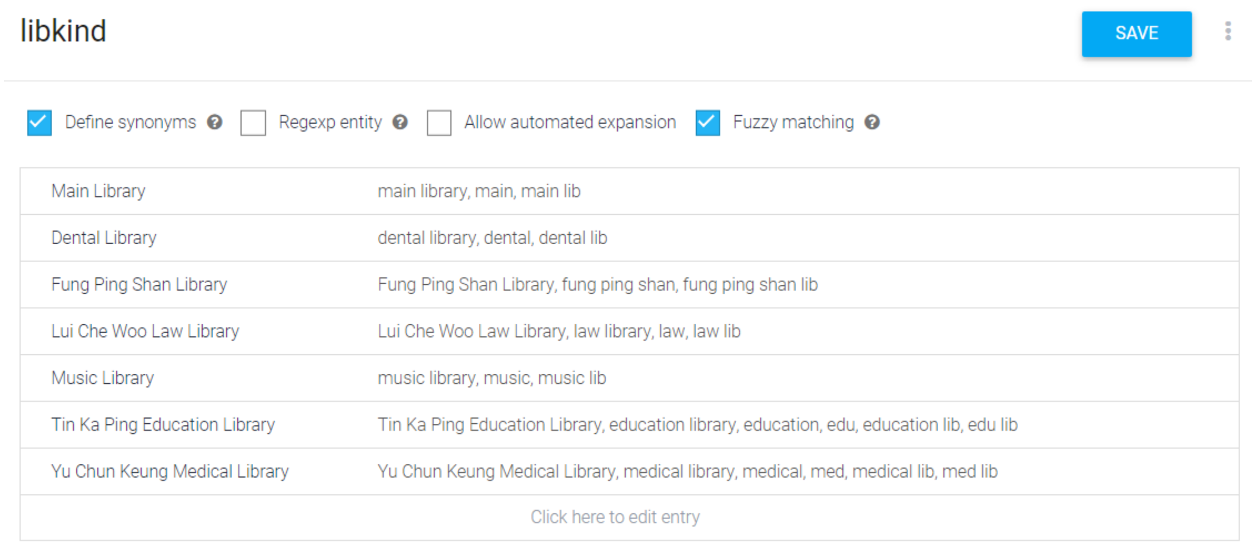
DEFAULT GOOGLE ASSISTANT +

Text Response

1 Sorry, I can only answer limited questions about the libraries. f you didn't specify your question, please rephrase. If not, please contact to the librarians directly: libcir@hku.hk

Figure II.7 Fallback intent

There were two default intents in Dialogflow, welcome and fallback intent. Welcome intent is the intent that is triggered when the users say greetings. Fallback intent is triggered when the model cannot confidently (over 50%) match the input phrase to any intent. By default, asking the users to rephrase the question was the response of fallback intent. There are three types of questions that can trigger the fallback intent in this project; 1) questions unrelated to the libraries, 2) unclear questions without enough details such as ‘main library’ instead of ‘opening hours of main library’ and 3) sophisticated questions that require librarians to answer. Therefore, the response to the default question was discussed with the client and rephrased as shown in Figure *II.7*.

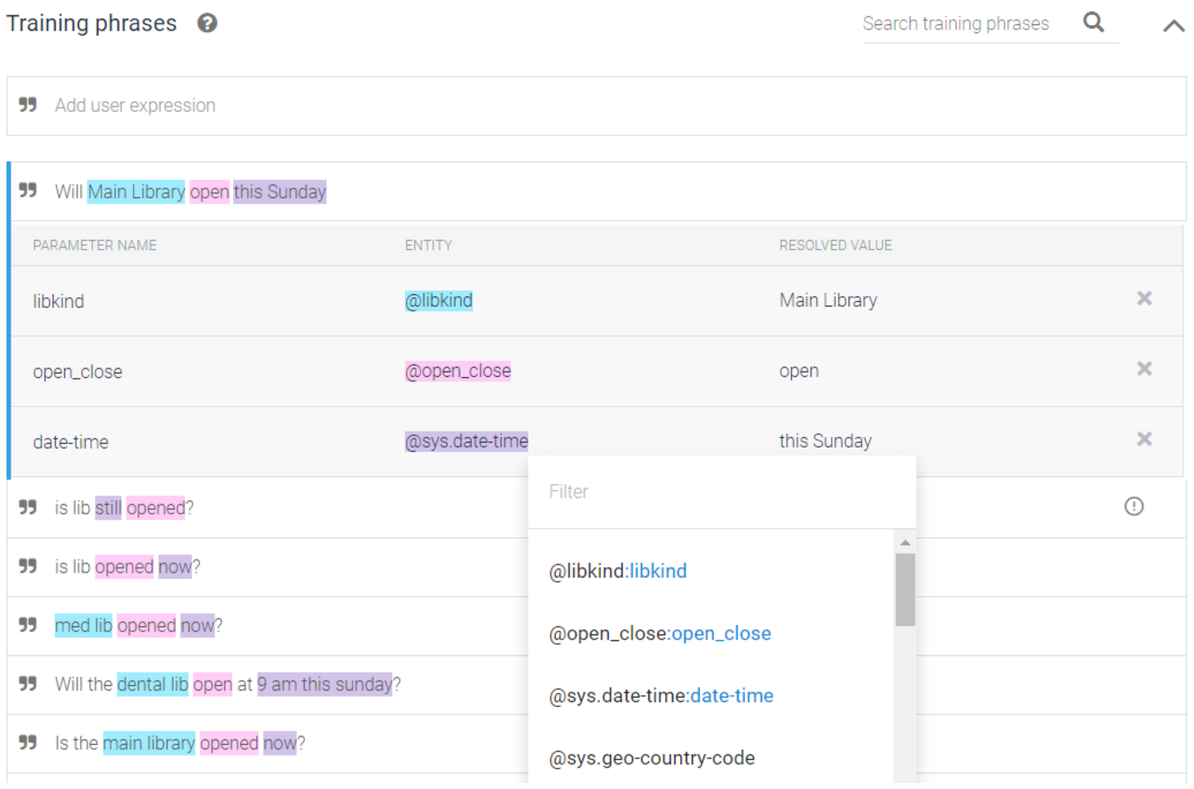


☑ Define synonyms ☐ Regexp entity ☐ Allow automated expansion ☑ Fuzzy matching

Main Library	main library, main, main lib
Dental Library	dental library, dental, dental lib
Fung Ping Shan Library	Fung Ping Shan Library, fung ping shan, fung ping shan lib
Lui Che Woo Law Library	Lui Che Woo Law Library, law library, law, law lib
Music Library	music library, music, music lib
Tin Ka Ping Education Library	Tin Ka Ping Education Library, education library, education, edu, education lib, edu lib
Yu Chun Keung Medical Library	Yu Chun Keung Medical Library, medical library, medical, med, medical lib, med lib

Figure II.8 Entity ‘libkind’ in Dialogflow

Before constructing further intents, entities were first defined since they should be added to intents. Each entity may have multiple values, and the entities would be used as parameters to find and call corresponding values from user inputs. There are reference value and synonyms for each entity value. Every synonym would be found from the input, and corresponding reference value would also be found by the model. For instance, when ‘where is dental lib?’ is the input question, ‘dental lib’ is found as the original value of ‘libkind’ entity, and the reference value ‘Dental Library’ is saved as the parameter value. The reference value would be used frequently in NLU, such as searching through the database as shown in Figure *II.4*. Therefore, the reference values should match to the keys in the database. ‘Fuzzy matching’ allows typos or some misspellings supported by Google Natural Language API.


Training phrases

Search training phrases

Add user expression

Will Main Library open this Sunday

PARAMETER NAME	ENTITY	RESOLVED VALUE
libkind	@libkind	Main Library
open_close	@open_close	open
date-time	@sys.date-time	this Sunday

is lib still opened?

is lib opened now?

med lib opened now?

Will the dental lib open at 9 am this sunday?

Is the main library opened now?

Filter

@libkind:libkind

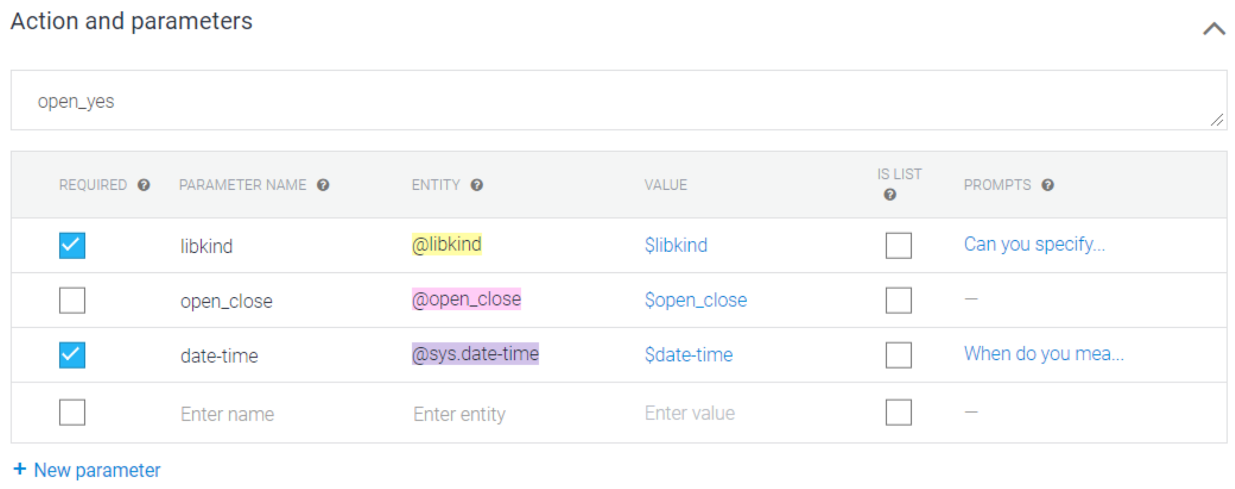
@open_close:open_close

@sys.date-time:date-time

@sys.geo-country-code

Figure II.9 Training phrases on Dialogflow Intents tab

There are many components that need to be defined in each intent; intent name, contexts, events, training phrases, parameters, responses and webhook call enablement. First, intent names matching the flowchart were set. Then, few training phrases were added. Highlighted word/s could be matched to parameters predefined in the entities tab as in Figure *II.9*. The model might find parameters automatically, and missed or mismatched parameters could be modified under 'Training phrases.'



Action and parameters

open_yes

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
✓	libkind	@libkind	$libkind	☐	Can you specify...
☐	open_close	@open_close	$open_close	☐	–
✓	date-time	@sys.date-time	$date-time	☐	When do you mea...
☐	Enter name	Enter entity	Enter value	☐	–

Figure II.10 Entities setting of intents

When values from the synonym list were found on the training phrases, they were added to parameters automatically. Any more entities could be added as the parameters of intents under ‘Action and parameters.’ ‘Required’ parameters are the mandatory values on the intent. When the model could not find required parameters from the original user question, one of questions on ‘prompts’ would be triggered, and the user response would be stored as the corresponding parameter value.

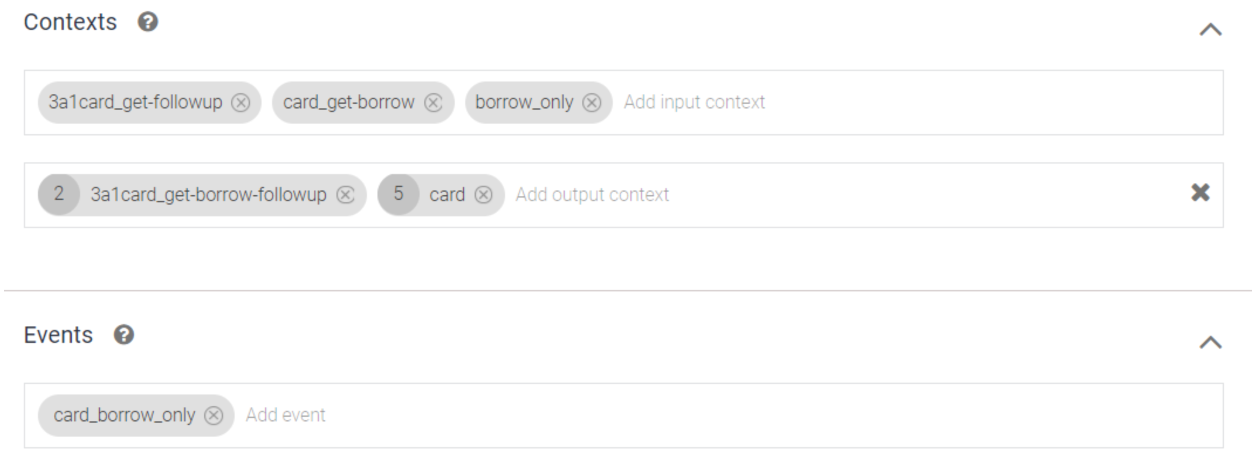
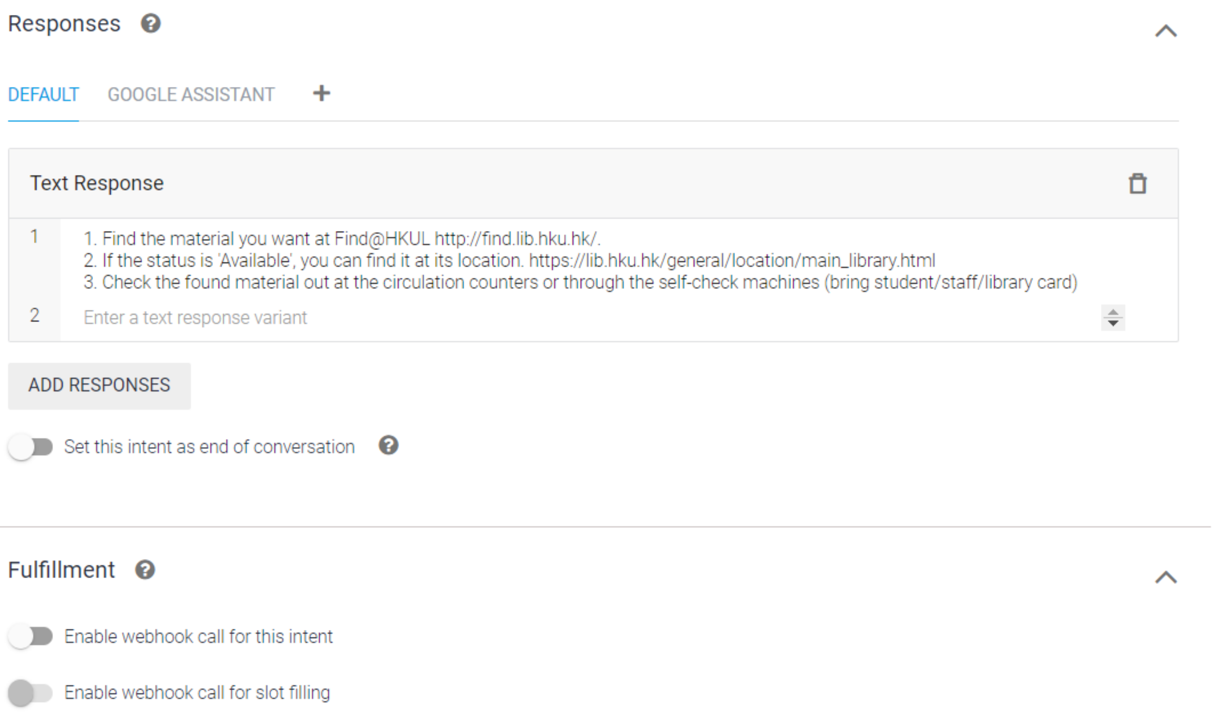


Figure II.11 Contexts and Events of intents

Contexts would be used for interactions between intents of card registration intents with the workflow shown in *Figure II.3*. When an intent is triggered by user input matching to trained phrases, its input contexts and output contexts will be triggered. When the next intent triggered has any of the previous output contexts as an input context, two intents have some connected features since contexts allow intents to share the parameter values. For example, *Figure II.11* shows the contexts of 'card_get – borrow' intent. Its parent intent 'card_get' output 'card_get-borrow' context, and when the intent was triggered, 'card_get-borrow' was triggered as input contexts as well. Any parameters found from 'card_get' intent was saved in the context 'card_get-borrow' and could be used in 'card_get – borrow' intent.

Events are used to trigger the intent manually in fulfillment by the function `webhookClient.setFollowupEvent(event_name)` [3]. Events were used for the intents that require some variations in the workflow. For example, there are intent A, B, the follow-up intent of A, and C, the follow-up intent of B. After the intent A, by some condition of the parameter value, the intent B should be triggered without input matching to trained phrases of B, or the intent C should be triggered while skipping the intent B. Such variation could be managed by the fulfillment code of the intent A with the event B or C.


Responses

Text Response

1. Find the material you want at Find@HKUL http://find.lib.hku.hk/.
2. If the status is 'Available', you can find it at its location. https://lib.hku.hk/general/location/main_library.html
3. Check the found material out at the circulation counters or through the self-check machines (bring student/staff/library card)

Enter a text response variant

Set this intent as end of conversation

Fulfillment

Enable webhook call for this intent

Enable webhook call for slot filling

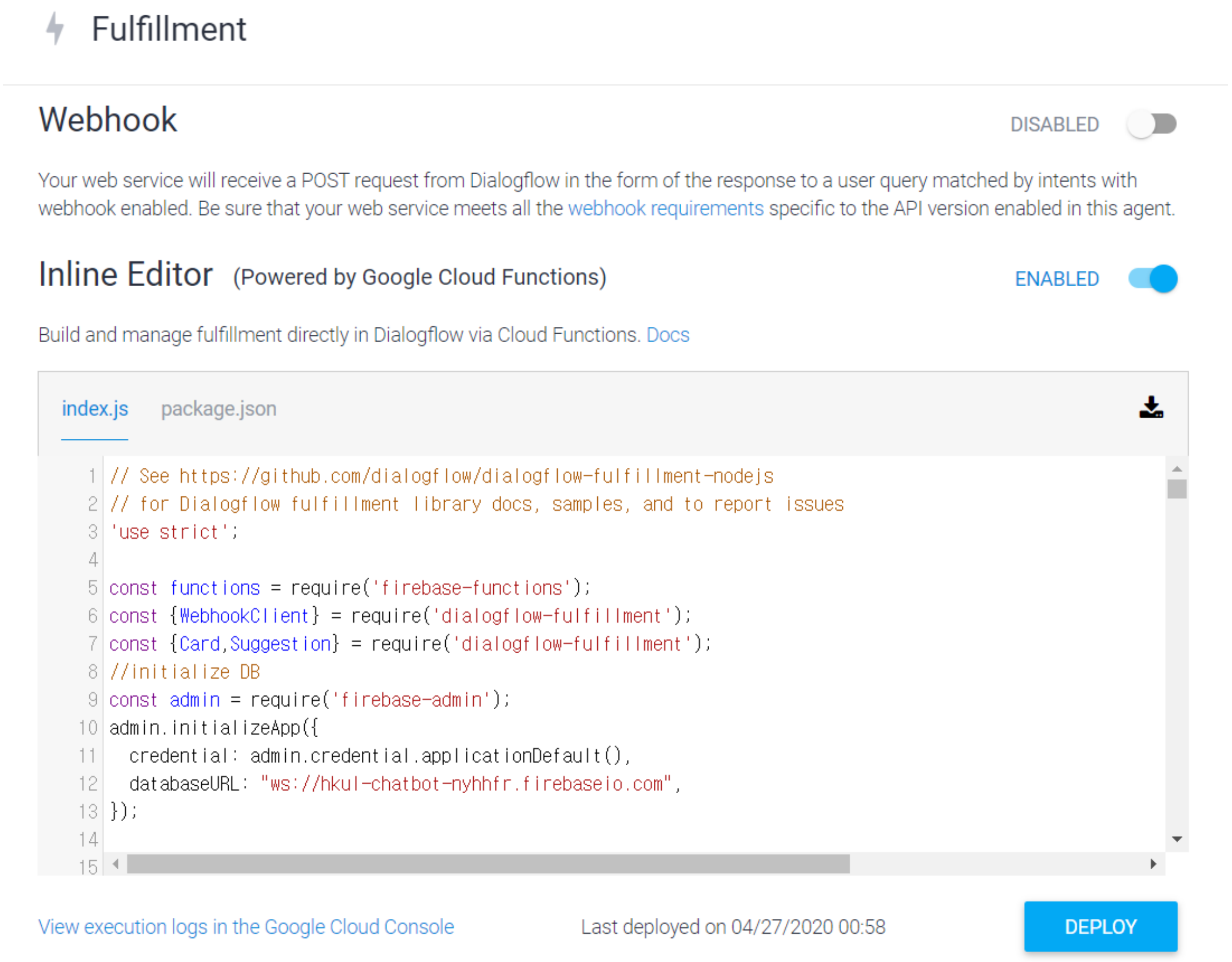
Figure II.12 Response setting of intents

After the model could understand the question, the answer to the question would be sent to the users. There are two types of responses used in the chatbot, static and dynamic responses. An intent with a static response always outputs the same response. There may be variations in static response when the information they send to user is same but the sentence structure or the arrangement of words varies. Such response can be set in the Intents tab on Dialogflow under 'Responses'. The model will randomly output any one of the 'Text Responses' when the intent is triggered. For greetings and farewell intents, multiple response variants would be added to give more human-like experience. Rest of intents providing information from the HKUL website would not require variant responses. Also, most of the informative responses would contain many data, which would not be easy to change the sentence structure dynamically.

The bot would finish the conversation when a user inputs farewell like 'bye'. For this function, farewell intent would be added, variants of farewells would be added to training phrases, and 'Set this intent as end of conversation' would be selected.

Responses that require information differ by entities would use the database, and such responses are called dynamic responses. To fetch different information from the database, webhook call should be enabled for such intents. Thus, when one of the intents is triggered, fulfillment code that gets data from database forms the response and outputs it.

II.4-2 Fulfillment



Fulfillment

Webhook DISABLED

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the webhook requirements specific to the API version enabled in this agent.

Inline Editor (Powered by Google Cloud Functions) ENABLED

Build and manage fulfillment directly in Dialogflow via Cloud Functions. Docs

index.js package.json

```
// See https://github.com/dialogflow/dialogflow-fulfillment-nodejs
// for Dialogflow fulfillment library docs, samples, and to report issues
'use strict';

const functions = require('firebase-functions');
const {WebhookClient} = require('dialogflow-fulfillment');
const {Card,Suggestion} = require('dialogflow-fulfillment');
//initialize DB
const admin = require('firebase-admin');
admin.initializeApp({
  credential: admin.credential.applicationDefault(),
  databaseURL: "ws://hkul-chatbot-nyhhfr.firebaseio.com",
});

```

View execution logs in the Google Cloud Console Last deployed on 04/27/2020 00:58 DEPLOY

Figure II.13 Fulfillment tab on Dialogflow

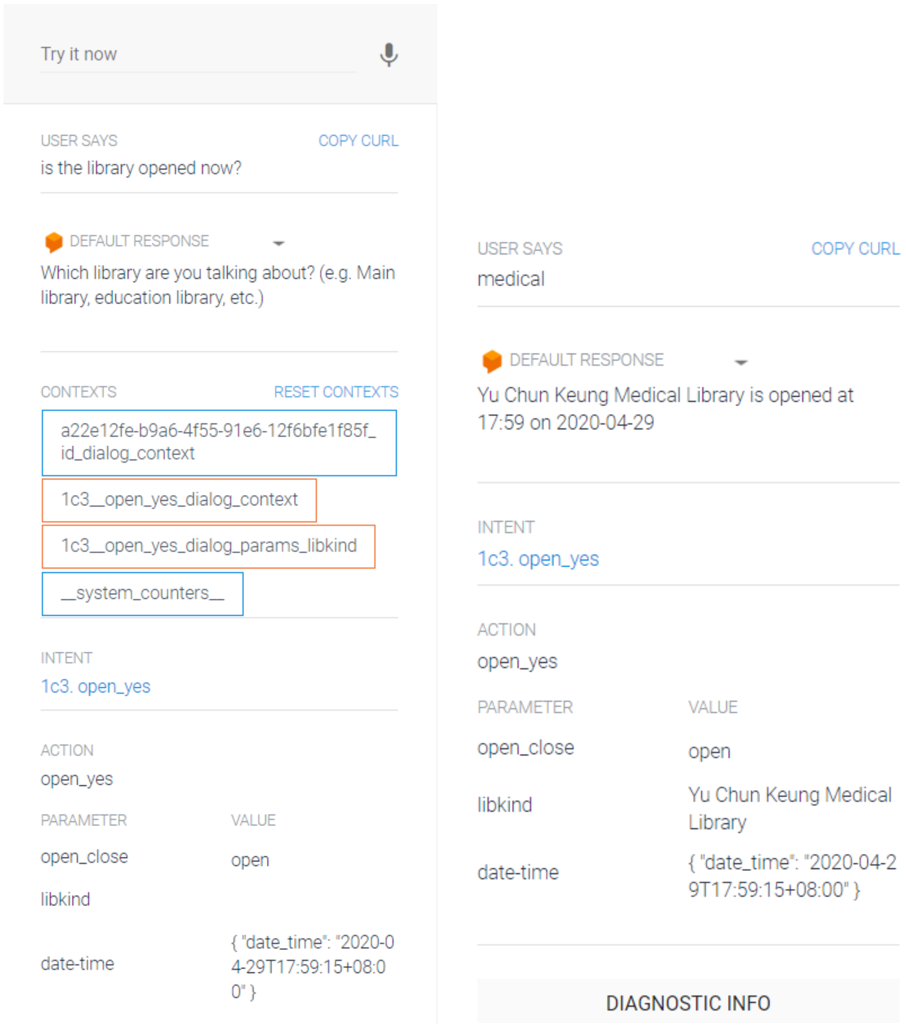
Fulfillment can be used locally with webhook, but Inline Editor on Dialogflow console was used for better collaborative work and easier transfer to the clients. index.js is the main function that runs when any of the intents enabled the webhook call is triggered from the NLU model. The default code from Dialogflow builds the basic setting. Besides it, the Firebase database needed to be established before further development as follow.

```
1       const admin = require('firebase-admin');
2       admin.initializeApp({
3         credential: admin.credential.applicationDefault(),
4         databaseURL: "ws://hkul-chatbot-nyhhfr.firebaseio.com",
5       });
```

Line 4 includes the link to Firebase database. ‘hkul-chatbot-nyhhfr’ is the project code on Firebase. The following code shows the basic format of functions of an intent.

```
1       intentMap.set(‘intent_name’, function_name);
2       agent.handleRequest(intentMap);
3
4       function function_name(agent){
5            entity1 = agent.parameters.parameter_name;
6            const Ref = admin.database().ref(‘key1/key2’);
7            return Ref.once(“value”).then((snapshot) => {
8                 value_from_db = snapshot.child(‘key3’).val();
9                 agent.add(‘response ${entity1}’);
10           });
11      }
```

The first two lines is used to run the corresponding function when an intent is triggered from the NLU model. `agent` is defined as `const agent = new WebhookClient({ request, response })` from the default code. Line 5 is the function to call the parameter values. It calls the reference value, and the original value used by the user can be called by adding `.original` after parameter_name. On line 6, Ref contains the database reference path which is used in line 7 to form the response by getting the data from the path. Line 8 fetches the data from database. As a result, it gets the value from the path `key1/key2/key3`, and `snapshot.key()` can fetch the keys of Ref. `agent.add(response)` will send the response to Dialogflow.



Try it now

USER SAYS COPY CURL
is the library opened now?

DEFAULT RESPONSE
Which library are you talking about? (e.g. Main library, education library, etc.)

CONTEXTS RESET CONTEXTS
a22e12fe-b9a6-4f55-91e6-12f6bfe1f85f_id_dialog_context
1c3__open_yes_dialog_context
1c3__open_yes_dialog_params_libkind
__system_counters__

INTENT
1c3. open_yes

ACTION
open_yes

PARAMETER	VALUE
open_close	open
libkind	
date-time	{ "date_time": "2020-04-29T17:59:15+08:00" }

USER SAYS COPY CURL
medical

DEFAULT RESPONSE
Yu Chun Keung Medical Library is opened at 17:59 on 2020-04-29

INTENT
1c3. open_yes

ACTION
open_yes

PARAMETER	VALUE
open_close	open
libkind	Yu Chun Keung Medical Library
date-time	{ "date_time": "2020-04-29T17:59:15+08:00" }

DIAGNOSTIC INFO

Figure II.14 Dialogflow testing console example

After the code was written on the inline editor, it should be deployed to save and update the model. To test and debug, log info with error message and information such as matching intent, found parameters, etc. was needed which were provided on the testing console of Dialogflow. As Figure *II.14* shows, testing console takes question, shows triggered intent and contexts, fetched parameter values, prompts of required parameters and responds. Diagnostic info shows the full JSON request and response. Log can be viewed from Google Cloud through link on the bottom of fulfillment tab shown in Figure *II.13*.

1

II.4-3 Model Training

After the model was built, more testing and training should be done to prevent possible errors and enhance the performance. Internally within the group members, Dialogflow testing console shown in Figure *II.14* could be used. However, external parties could not and should not have access to the model directly. Hence, the web demo from Dialogflow was used.

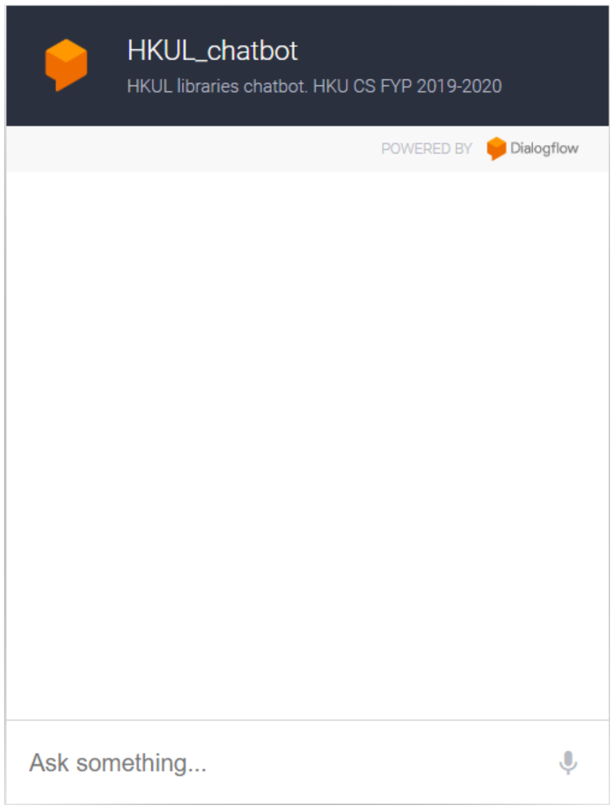
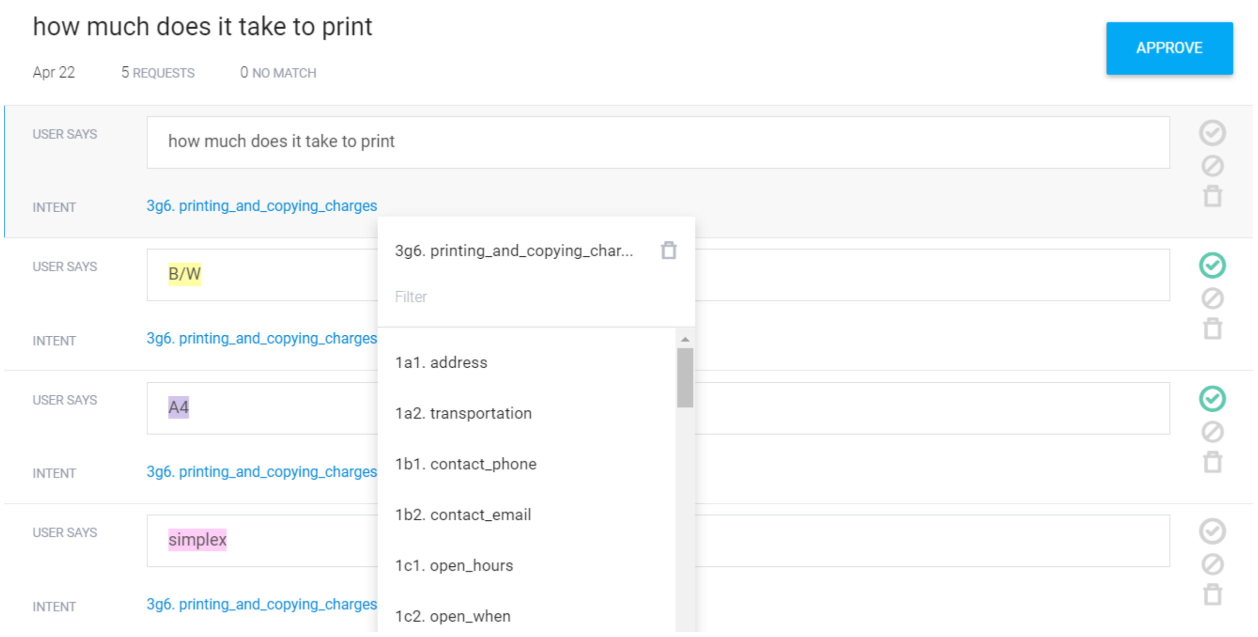


Figure II.15 Web Demo UI

It is accessible by the link https://bot.dialogflow.com/hkulchatbot. Although it only demonstrates the minimum functions and does not even support line change on the response, it could be used in meetings, presentations and communication with the clients. All conversations through web demo would be saved in the log history and Dialogflow training which will be introduced next.


how much does it take to print

Apr 22 5 REQUESTS 0 NO MATCH

USER SAYS: how much does it take to print
INTENT: 3g6. printing_and_copying_charges

USER SAYS: B/W
INTENT: 3g6. printing_and_copying_charges

USER SAYS: A4
INTENT: 3g6. printing_and_copying_charges

USER SAYS: simplex
INTENT: 3g6. printing_and_copying_charges

Figure II.16 Training tab example on Dialogflow

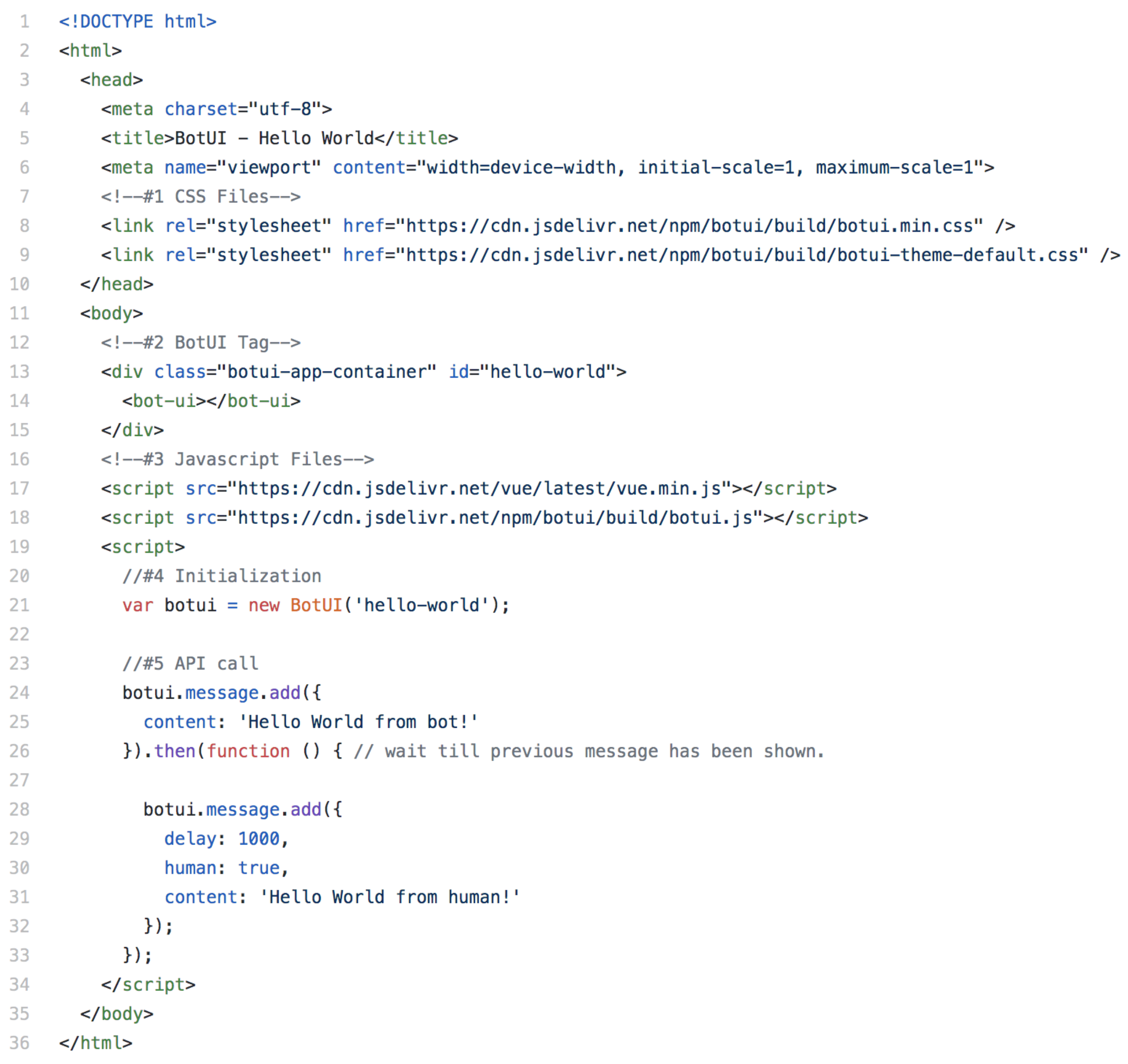
Not only the logs on the Google Cloud, but training tab on Dialogflow also records the history of the NLU model. It shows the matched intents and fetched parameters by every input. If the intent or some parameter values are mismatched or some parameter values are missing, observable from the diagnostic info from Figure *II.14*, training tab allows modification; changing the matched intent or parameter name, or selecting different values as parameters. Training allows more thorough NLU model training than from the training phrases of each intent since it shows all the input after prompts while training phrases only include the first question that can trigger the intent. The response to the prompt questions may be matched to different intent from the original question. Such error also can be fixed since training tab allows to change intents for every user input, including the user responses to prompt questions.

II.5 UI/UX

The chatbot was originally planned to be integrated to the HKUL official website. However, due to the COVID-19 quarantine, accessing the HKUL website server could not be done. The clients have access to web demo already, and suggestions for future integration will be included in the future documentation with maintenance instruction after the completion of the project. The librarians are well informed about it and considering alternatives such as to suggest it as separate project in the future. Since all process had been done on the web consoles, HKUL can access to all the tools by the Google account that has been used.

The UI tool that will be included to the documentation is BotUI, the Vue-based JavaScript framework for conversational programs. It is free and it has an intuitive API to add messages and show the possible actions from users. Four files need to be needed for setup and installation of BotUI:

- CSS file for the basic layout
- CSS file defining the overall look
- Vue.js framework
- JavaScript file for the BotUI framework



```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="utf-8">
5       <title>BotUI - Hello World</title>
6       <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
7       <!--#1 CSS Files-->
8       <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/botui/build/botui.min.css" />
9       <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/botui/build/botui-theme-default.css" />
10    </head>
11    <body>
12      <!--#2 BotUI Tag-->
13      <div class="botui-app-container" id="hello-world">
14        <bot-ui></bot-ui>
15      </div>
16      <!--#3 Javascript Files-->
17      <script src="https://cdn.jsdelivr.net/vue/latest/vue.min.js"></script>
18      <script src="https://cdn.jsdelivr.net/npm/botui/build/botui.js"></script>
19      <script>
20        //#4 Initialization
21        var botui = new BotUI('hello-world');
22
23        //#5 API call
24        botui.message.add({
25          content: 'Hello World from bot!'
26        }).then(function () { // wait till previous message has been shown.
27
28          botui.message.add({
29            delay: 1000,
30            human: true,
31            content: 'Hello World from human!'
32          });
33        });
34      </script>
35    </body>
36  </html>
```

Figure II.17 Hello World chatbot HTML

After the setup, HTML file will be created. Figure *II.17* is the sample HTML file of Hello World chatbot. As shown, the basic theme and layout CSS should be included inside `<head>` tag. Inside `<body>`, there are `<div>` that creates the container to hold the chatbot UI and JavaScript files. Global variables can be created which can be initialized with `id` of the `<bot-ui>` tag as the parameter. Messages can be added to the bot by calling the API as shown in line 31.

Some API calls implemented in the HTML file allows to create an interactive chatbot. For example, ‘message’ displays messages on the UI;

- `message.bot` displays the messages output from the chatbot
- `message.human` displays the messages input by the users
- `message.add` displays the messages in general

‘markup’ is used to display diverse data besides text, such as images, icons and links without manual modification on HTML. ‘action’ provides different options to respond to users such as typing text directly by `action.text`, multiple choices by `action.button` or dropdown lists to select one or more by `action.select`.

III. RESULT

Construction of the NLU model and the database started from November 2019. This section explains how the process went on, what difficulties were faced and how they were solved, and what changes had been done from the initial plan. Lee designed workflow, constructed the NLU model for intents with dynamic responses and some of database while Sung built the rest of the database and Kong built most of the NLU model for intents with static responses. Every member trained the model by adding more phrases, and Lee debugged. Sung and Kong researched for UI and UX.

III.1 Flowchart

Some minor edits were made on the flowchart for better workflow. For example, initially, no user type intents were included to the primary intent 'card_get' as parameters, and the intent asked for the user type as the default text response on Intents tab and the follow-up intents were triggered according to the response after. However, by the suggestion from the supervisor, users may mention their user type already in the first question. Hence, all possible user type entities of the follow up intents were added to 'card_get' but not as required parameters, and the code to trigger whether the follow-up intents to get the parameter value or skip it and trigger the next follow-up intents was written in fulfillment. Flowchart was also edited to reflect such changes shown by the arrows on Figure *II.3*. Besides, some minor changes such as letter case were made on the flowchart while the project proceeded.

Overall, there are 7 types of questions of the basic information of the libraries, 12 questions of the user policies, and 37 questions of facilities. All 7 questions in scope 1 output the dynamic responses, while 6 in scope 2 and 9 in scope 3 are connected to webhook and form dynamic responses.

The fourth scope of document searching from the HKUL API was excluded from the scope after the interim presentation due to the budget and billing information issue. HKUL API should be called in the Dialogflow fulfillment for document searching function on the chatbot. However, Google Cloud requires the Blaze Plan for external API connections which allows some free tier but requires the billing information. However, the client did not have budget or credit card information allocated for this project. The librarians suggested to use any personal credit card from one of the members just for this project, but the members decided that it is not necessary to add functions that cannot be used by the client in the future. The clients were informed about it and agreed.

III.2 Database

Most of the structure followed the flowchart, but some changes were made along with developing the NLU model and fulfillment codes for agreement between all of them. The database should be constructed before testing the corresponding codes, but since database construction was assigned to different member to the one writing the codes, the database was sometimes not ready for testing. Disagreement mostly happened in such circumstances. Errors not caused by such issues were not easily discernible since the values defined in the NLU model needed to be compared to the keys and values in the database, while the codes were flawless, and the execution log only shows the lines of the fulfillment codes causing the errors. In this case, the parameter values were null since it cannot find such path in the database. Therefore, the database values should be compared to the entity reference values, intent names and parameter values before debugging.

Format of some data also had been changed due to workflow. For example, the opening hours of the libraries first built in the format ‘10:00 am’ whereas time on sys.date-time of Dialogflow is ‘10:00:00’. Comparison between two were needed in some intents and system entity values cannot be modified, and thence time data on the database were modified to the format of Dialogflow system entities.

There had been one concern that the opening hours of the branches may change, like due to current quarantine. To make the database to self-update, it needs to run the web scraping program on the HKUL website or HKUL google information automatically at least once a week. This aspect was discussed with the clients, and it was concluded that HKUL will update the Firebase database directly or find for better solution in future projects. The instructions to modify data from the database including write operations on web for automation will be described in the documentation.

III.3 NLU model

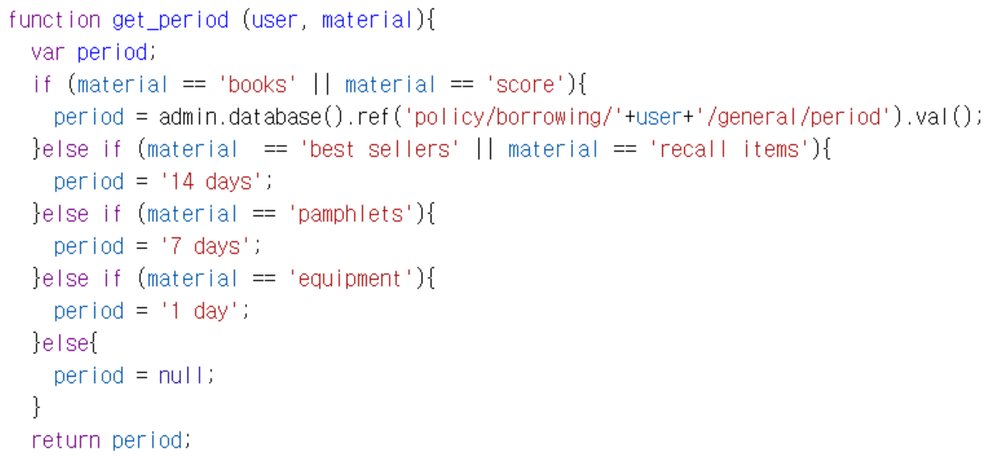
NLU models were built according to the flowchart and some revision on the flowchart also had been made when needed as section III.1 Flowchart showed. Revision on workflow of ‘card_get’ intents caused some adjustments. Some user types that are applied by the same policy or only one variation had not been needed to be called as parameters, but the primary intent asks for user type and the follow-up intent needed to be trained with such user types. For example, all alumni are considered same type of user without any variation. Therefore, ‘card_get – HKUgrad’ had been trained with synonyms of alumni only without any parameters defined. On the other hand, full time staffs and students may apply to temporary extensions. Although, the extension process is same to all of them, so the NLU model did not need to have HKUfull entity with reference values ‘full-time staff’ and ‘students’. These values were added to the training phrases of ‘card_get – HKUfull’ intent which had no parameters either. However, to fetch any of user types from ‘card_get’ intent, all possible user types that may trigger the follow-up intents were created as entities. ‘card_get’ intent now have five parameters, ‘borrow_only’, ‘external’, ‘HKUfull’, ‘card_rest’ and ‘HKUgrad’, and each follow-up intents have corresponding entities as parameters.

After all the workflow had been verified to be successfully implemented, testing had been done. By the training tab on Dialogflow, incorrect matching of intents or missed parameter values were found and corrected. This method to further improve the NLU model had been successfully done after the last meeting with the libraries when the demo was introduced. By many library officials try the demo, various questions were collected, and the model was retrained from them.

Effort to enhance the user experience of interactive chatbot added farewell intent and redesigned the response on the fallback intent. Farewell intent was added to let users to end and restart the conversation more intuitively. Fallback intent is important for user experience since the objective of the chatbot is only limited, and many of inquiries may not be answered by the chatbot. Therefore, information needed to be provided to users when they could not get the answers to their inquiries was discussed with HKUL and the email address HKUL requested was added to the response to the fallback intent.

III.4 Fulfillment

Fulfillment was the most time consuming and important process in the project since the static responses did not require any complex logic to accomplish, and fulfillment mediates the NLU model and database to construct dynamic responses. This section introduces additional modifications that had been done to improve the chatbot.



```
function get_period (user, material){
  var period;
  if (material == 'books' || material == 'score'){
    period = admin.database().ref('policy/borrowing/'+user+'/general/period').val();
  }else if (material  == 'best sellers' || material == 'recall items'){
    period = '14 days';
  }else if (material == 'pamphlets'){
    period = '7 days';
  }else if (material == 'equipment'){
    period = '1 day';
  }else{
    period = null;
  }
  return period;
```

Figure III.1 get_period function

First, most of the functions in the fulfillment are intent functions, but from the nature of information, function get_period was defined. According to the HKUL policy, loan period of material depends on the user and material types and affects other policies such as the maximum loan renewal period, the waiting time of holds and overdue fine. Moreover, the loan periods of some materials such as pamphlets, leisure books and equipment are same for all the user types. Therefore, get_period function was defined for the efficient code, and only the information for normal books which differs by the user types was added to the database while the common values are given in get_period code directly.

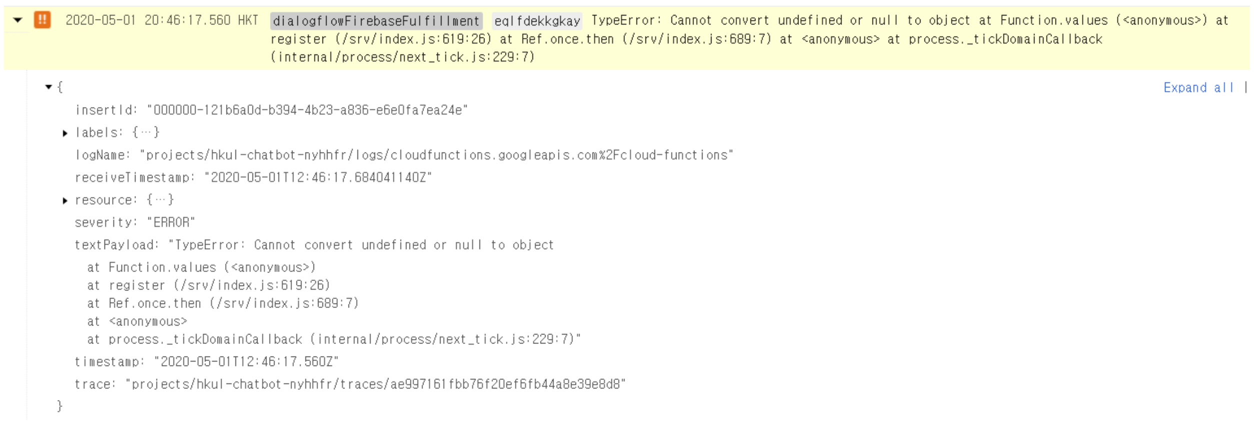
Second, some intents were artificially triggered in the fulfillment code without user input and parameter values were shared among the intents. Triggering intent could be done by `webhookClient.setFollowupEvent(event_name)` introduced in section II.4-2 Fulfillment. Then, it was planned to use the parameter values stored in the contexts. However, event triggers the intents from the outside of the conversation. In other words, contexts lose its parameter values when event triggers an intent. However, `setFollowupEvent` also could be passed with parameter values. Therefore, the intents that needed to be triggered by event had been called by

```
agent.setFollowupEvent({‘name’: ‘event_name’,
                        ‘parameters’:{‘param1’: ‘parameter1’}});
```

The code above changes the parameter of the triggered intent directly, so the values could be called as the parameter values fetched from the user question directly. Still, intents asking for the card types are triggered by context. Parameters from contexts had been called differently since the triggered intent does not contain the parameter values directly but the contexts do in this case.

```
const parent = agent.context.get('input_context');
var user = parent.parameters.param1;
```

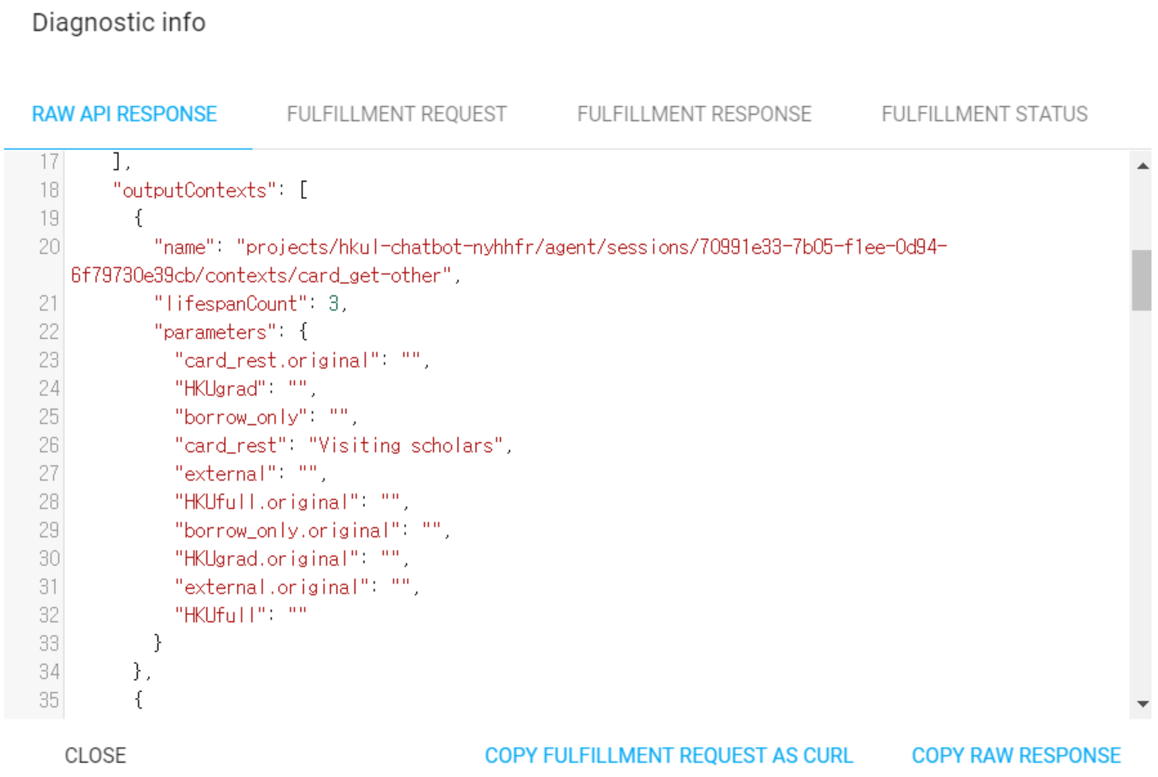
The code above is the method to get the parameters from the previous intent through context. These updates had been done through debugging.



```
2020-05-01 20:46:17.560 HKT dialogflowFirebaseFulfillment eqlfdekkgkay TypeError: Cannot convert undefined or null to object at Function.values (<anonymous>) at
                            register (/srv/index.js:619:26) at Ref.once.then (/srv/index.js:689:7) at <anonymous> at process._tickDomainCallback
                            (internal/process/next_tick.js:229:7)
{                                                                                                                                   Expand all |
  insertId: "000000-121b6a0d-b394-4b23-a836-e6e0fa7ea24e"
  labels: {…}
  logName: "projects/hkul-chatbot-nyhhfr/logs/cloudfunctions.googleapis.com%2Fcloud-functions"
  receiveTimestamp: "2020-05-01T12:46:17.684041140Z"
  resource: {…}
  severity: "ERROR"
  textPayload: "TypeError: Cannot convert undefined or null to object
    at Function.values (<anonymous>)
    at register (/srv/index.js:619:26)
    at Ref.once.then (/srv/index.js:689:7)
    at <anonymous>
    at process._tickDomainCallback (internal/process/next_tick.js:229:7)"
  timestamp: "2020-05-01T12:46:17.560Z"
  trace: "projects/hkul-chatbot-nyhhfr/traces/ae997161fbb76f20ef6fb44a8e39e8d8"
}
```

Figure III.2 Google Cloud error logs

Debugging the fulfillment code relied on the Google Cloud logs to position the error lines and the details of error. Figure *III.2* shows the error caused by defining the database path with a null value.



```
  ],
  "outputContexts": [
    {
      "name": "projects/hkul-chatbot-nyhhfr/agent/sessions/70991e33-7b05-f1ee-0d94-
6f79730e39cb/contexts/card_get-other",
      "lifespanCount": 3,
      "parameters": {
        "card_rest.original": "",
        "HKUgrad": "",
        "borrow_only": "",
        "card_rest": "Visiting scholars",
        "external": "",
        "HKUfull.original": "",
        "borrow_only.original": "",
        "HKUgrad.original": "",
        "external.original": "",
        "HKUfull": ""
      }
    },
    {
```

Figure III.3 API response information

API response had also been used for debugging, especially to check whether parameter values were fetched by the NLU model or they were able to be called by contexts or event. As shown in Figure III.3, parameter values of each intent could had been checked by diagnostic info from Dialogflow testing console.

Small updates also had been made for the response structures after all the necessary information were collected. Originally, it was not planned to add variations in dynamic responses, but the intents of basic information of library branches had short responses, variations were possible to apply and it would make the chatbot more appealing.

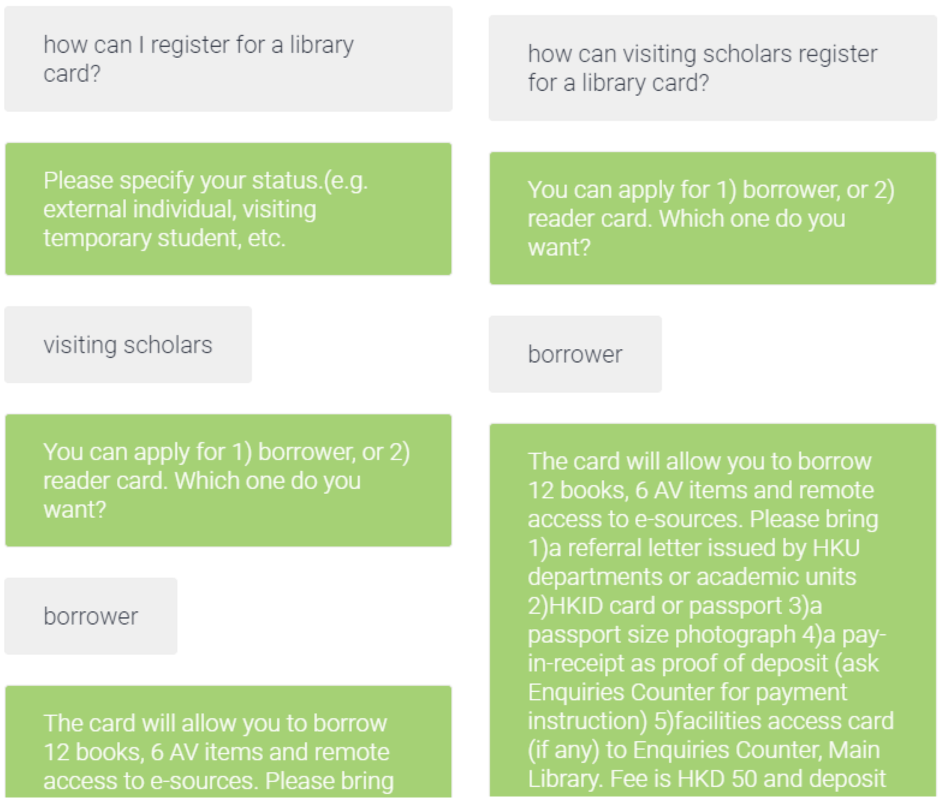
```
let response = [variation, variation2, …];
const randomRes = response[Math.floor(Math.random() *
response.length)];
```

List of variations were added to `response` and one from it was randomly extracted to `randomRes` which is the value output to Dialogflow as the final response.

III.5 Testing

To test the correctness of response and possibilities of bugs in fulfillment, testing had been done several times. Most of errors were caused by inconsistence in entity values or intent names as already discussed in section III.2 Database. Because some tools being used are case sensitive, disagreement on Dialogflow NLU model, database keys or the codes caused error on the fulfillment execution. Such error might not be found during the primary testing after deploying the codes since the parameter values used in the testing may not have issues while other parameter of the same entity has. Therefore, the secondary testing should be done using phrases with diverse parameter values.

Another common error was caused by intent matching. Even though Dialogflow uses the Google Natural Language API, some patterns may not be matched to the corresponding intent when similar phrases were not trained. Therefore, every time the question that should be able to be answered by the chatbot was mismatched or not matched to any intent and fallback intent was triggered, it was adjusted to the correct intent by training tab as shown in Figure *II.16* and the NLU model was retrained. Moreover, sometimes entities were not found even though the input question had such information. In such cases, similar to intent matching, corresponding information was modified on training, and the model was retrained.



how can I register for a library card?

Please specify your status.(e.g. external individual, visiting temporary student, etc.

visiting scholars

You can apply for 1) borrower, or 2) reader card. Which one do you want?

borrower

The card will allow you to borrow 12 books, 6 AV items and remote access to e-sources. Please bring

how can visiting scholars register for a library card?

You can apply for 1) borrower, or 2) reader card. Which one do you want?

borrower

The card will allow you to borrow 12 books, 6 AV items and remote access to e-sources. Please bring 1)a referral letter issued by HKU departments or academic units 2)HKID card or passport 3)a passport size photograph 4)a pay-in-receipt as proof of deposit (ask Enquiries Counter for payment instruction) 5)facilities access card (if any) to Enquiries Counter, Main Library. Fee is HKD 50 and deposit

Figure III.4 Card registration of user type ‘others’ testing example

Figure *III.4* shows one successful testing as an example of card registration inquiry. User type ‘others’ is used as an example since it has the one of the most complicated workflows. The first question of both conversation triggers ‘card_get’ intent. On the conversation on the left, the first user question does not include the user type parameter. Therefore, question asking for the information is sent, and the user response ‘visiting scholars’ triggers the follow up intent ‘card_get – others’ by contexts. The question asking for card type is the response of ‘card_get – others’ intent, and user response ‘borrower’ triggers the follow up intent ‘card_get – others – borrower’. On the other hand, the conversation on the right can fetch the user type parameter from the first question. Therefore, the event triggers ‘card_get – others’ intent without contexts. From then, it follows the same workflow of the conversation on the left.

Testing had been done for several times for each intent, but because of the characteristics of human language, mismatching of intent and missing the corresponding parameter values may happen in the future again. Concerning such issues, instruction to retrain the model by such errors will be included in the documentation for the clients for future maintenance.

IV. Conclusion

The objective of the project was to develop an interactive chatbot that can answer to HKUL user enquiries. Using Dialogflow, the chatbot that can respond to most of the enquiries within the discussed scopes with the clients was developed. Although the integration on the website could not be done during this challenging period, all the essential aspects of the chatbot had already been accomplished. The chatbot can engage with the user by asking questions necessary to provide more accurate information by prompts and contexts in Dialogflow NLU model. The questions expecting the responses differ by information the user provided can fetch corresponding information from the Firebase database and structure the response according to the fetched data.

The NLU model performance was satisfying, but there are still chances of misclassification. By the characteristics of ML, more user input phrases and retraining the NLU model from those inputs will improve the accuracy of the chatbot. Therefore, future maintenance is as important, and the documentation for HKUL will explain the process in detail. In addition, any further scope expansion can be easily done by the clients for their own needs since the tools that do not require complicated migration were selected. The documentation will also briefly illustrate the methodologies used in the project for their reference.

References

[1] The University of Hong Kong Libraries, "The University of Hong Kong Libraries," 7 2019. [Online]. Available: https://lib.hku.hk/general/abouthkul/index.html. [Accessed 19 9 2019].

[2] J. Thanaki, in *Python natural language processing*, Birmingham, UK, Packt Publishing, 2019, pp. 372-373.

[3] "Dialogflow Fulfillment Library," Dialogflow, 25 9 2018. [Online]. Available: https://dialogflow.com/docs/reference/fulfillment-library/webhook-client.