S.A.N.A. Chatbot
Smart Academic Network Assistant

Group 6 team members
IAN TAN ENG KIONG
RANA BHATTACHARJEE
YEO WHYE CHUNG NELSON

Table of Contents

1 EXECUTIVE SUMMARY

In this project, the team developed a proof of concept (PoC) for a chatbot assistant to communicate with potential students of courses offered by the NUS Institute of Systems Science (NUS-ISS) via the Google Assistant platform.

Why does NUS-ISS need chatbots? User experience is of critical importance in today's society. Hence, it is essential to create superior customer experience for potential students in finding the right course at a faster speed. Also, awareness of chatbots and interest in the capabilities of this technology is growing among consumers today. One study [Brandtzaeg and Følstad 2017] identified the main factors motivating people to interact with chatbots:

- **Productivity.** Chatbots provide the assistance or access to information quickly and efficiently.
- **Entertainment.** Chatbots amuse people by giving them funny tips, they also help killing time when users have nothing to do.
- **Social and relational factors.** Chatbots fuel conversions and enhance social experiences. Chatting with bots also helps to avoid loneliness, gives a chance to talk without being judged and improves conversational skills.
- **Curiosity.** The novelty of chatbots sparks curiosity. People want to explore their abilities and to try something new.

ISS offers a large range of courses under different programmes. For this PoC, the scope of the chatbot is limited to answering questions about courses offered under the Executive Education programme. This scope covers over 100 course, organized under the following disciplines:

1) Artificial Intelligence	2) Cybersecurity
3) Data Science	4) Digital Agility
5) Digital Innovation & Design	6) Digital Products & Platforms
7) Digital Strategy and Leadership	8) Software Systems
9) Stackup – Startup Tech Talent Development	

Primarily, this chatbot is designed to answer queries on the following topics about individual courses:

1) Course overview	2) Upcoming classes
3) Key takeaways	4) Who should attend
5) What will be covered	6) Course fees
7) Funding schemes	8) Course instructors
9) Certification	10) Course preparation

This PoC will provide a good opportunity for NUS-ISS to explore the benefits of having a chatbot while identifying gaps to be addressed and future enhancements to potentially expand on the path towards a fully-developed chatbot in the future.

2 PROBLEM OVERVIEW

2.1 PROBLEM STATEMENT

Presently, there is no chatbot embedded within the existing NUS-ISS website. NUS-ISS offers a large range of courses under Executive Education, Graduate Programmes and Stackable Programmes with over 100 courses in Executive Education alone. Browsing through the many webpages of these courses to find out and compare various pieces of information would be a tedious and cumbersome process.

The current website does not provide a quick and efficient means to access such information, as each course is presented as a single static webpage. Given the IT-oriented nature of courses offered by NUS-ISS, it is imperative for NUS-ISS to adopt an intelligent chatbot providing a useful and interactive user experience. This is an important marketing and branding strategy for NUS-ISS as a prominent institution driving Singapore's future economy as a smart nation.

2.2 PROJECT OBJECTIVES

In this project, the team developed a PoC for a chatbot capable of answering queries from prospective students on courses offered under the Executive Education programme, providing targeted answers on individual courses queried. Through interaction with this chatbot, prospective students can solicit answers pertaining to each of the offered courses as well as answer certain more generic questions such as the list of courses offered under specific disciplines. The objective is to deliver a quick, effective, engaging and positive experience to prospective students when enquiring about courses by NUS-ISS.

2.3 PRODUCT OVERVIEW

The application is built using Google's Dialogflow technology, integrated with a Python webhook to provide dynamic responses. Dialogflow provides pre-defined entities for categorising extracted parameters, while allowing the creation of customised entities. The agent will match a user utterance with an intent, extracts identified parameters, and returns a response which can be either static or dynamic. This particular agent is designed to answer queries pertaining to courses in the Executive Education programme.

2.4 REQUIREMENTS & CONSTRAINTS

The application has the following inputs and features:

1) A set of intents pertaining to the courses
2) A set of customized entities
3) A Python script as the webhook for dynamic responses
4) Relevant organized data (scraped from the NUS-ISS website) pertaining to potential queries pertaining to the courses.
5) Google Assistant for testing the chatbot

3 TECHNICAL DISCUSSION

3.1 INTRODUCTION

This chatbot was built using Google's Dialogflow technology. Dialogflow requires a Google account for log in. If you do not have a Google account, you can get one here with your current email or sign up for a Google account and email at the same time with Gmail. There are 3 key components need by the Dialogflow platform:

Agent: A module within Dialogflow which incorporates Natural Language Processing to understand user utterances and identify which "action" to carry out. The agent transforms the user request into actionable data.

In this project, the agent is designed to answer queries pertaining to disciplines of the courses, individual courses and preparation information. It targets the 100+ courses in the Executive Education Programme where the information or knowledge can be represented as follows:

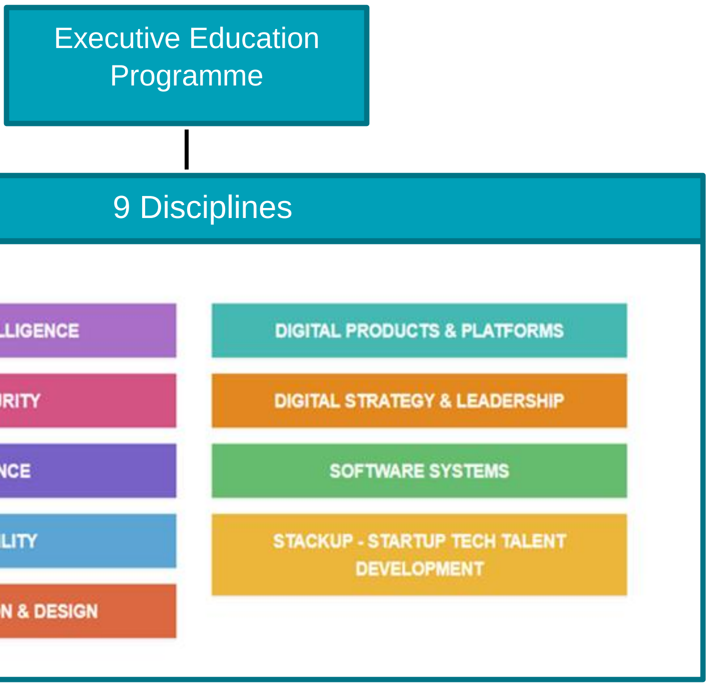
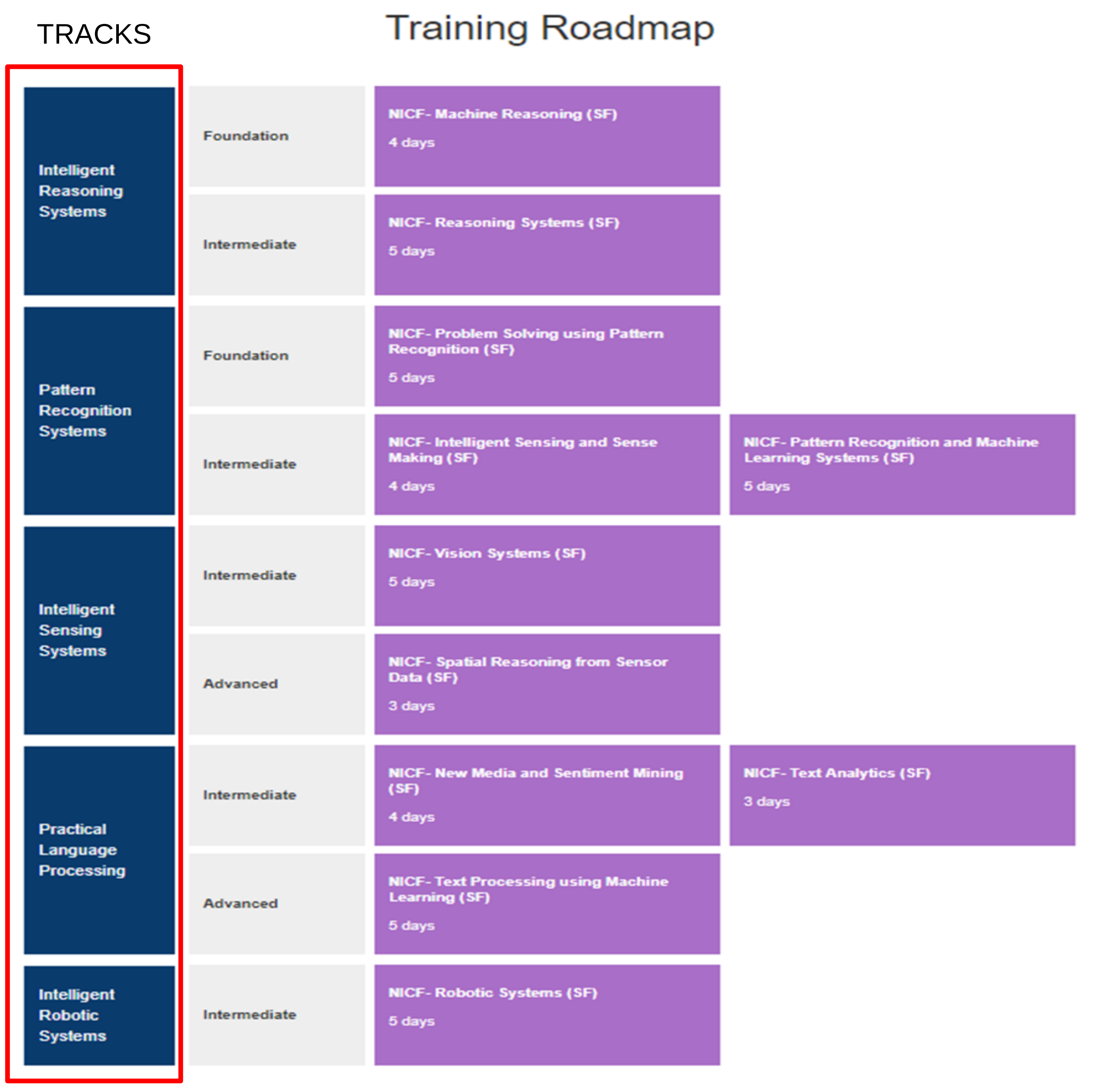

Executive Education Programme
9 Disciplines
ARTIFICIAL INTELLIGENCE
CYBERSECURITY
DATA SCIENCE
DIGITAL AGILITY
DIGITAL INNOVATION & DESIGN
DIGITAL PRODUCTS & PLATFORMS
DIGITAL STRATEGY & LEADERSHIP
SOFTWARE SYSTEMS
STACKUP - STARTUP TECH TALENT DEVELOPMENT

Figure 1: Disciplines under the Executive Education Programme

In each of the above disciplines, there are training roadmaps where a series of tracks are defined to provide the breakdown of courses connected to the track. An example of the training roadmap is as illustrated below:



TRACKS

Training Roadmap

Intelligent Reasoning Systems	Foundation	NICF- Machine Reasoning (SF) 4 days	
	Intermediate	NICF- Reasoning Systems (SF) 5 days	
Pattern Recognition Systems	Foundation	NICF- Problem Solving using Pattern Recognition (SF) 5 days	
	Intermediate	NICF- Intelligent Sensing and Sense Making (SF) 4 days	NICF- Pattern Recognition and Machine Learning Systems (SF) 5 days
Intelligent Sensing Systems	Intermediate	NICF- Vision Systems (SF) 5 days	
	Advanced	NICF- Spatial Reasoning from Sensor Data (SF) 3 days	
Practical Language Processing	Intermediate	NICF- New Media and Sentiment Mining (SF) 4 days	NICF- Text Analytics (SF) 3 days
	Advanced	NICF- Text Processing using Machine Learning (SF) 5 days	
Intelligent Robotic Systems	Intermediate	NICF- Robotic Systems (SF) 5 days	

Figure 2: The training road map for the Artificial Intelligence Roadmap

Overall there are more than 100+ courses in the assigned to the respective 9 disciplines. Also, there are 34 tracks identified from the 9 disciplines.

Intent: Support or the service that the user wants from the agent. Intent is configured by the developers. Intent determines the action by the code. In this project the following intents were created in DialogFlow.

1. GetCertification-hascontext	2. GetCertification-newcourse
3. GetCurriculum-hascontext	4. GetCurriculum-newcourse
5. GetEntityInfo	6. GetFees-hascontext
7. GetFees-newcourse	8. GetFundingSchemes
9. GetInstructors-hascontext	10. GetInstructors-newcourse
11. GetOverview-hascontext	12. GetOverview-newcourse
13. GetPrepInfo	14. GetTakeaways-hascontext
15. GetTakeaways-newcourse	16. GetTargetAudience-hascontext
17. GetTargetAudience-newcourse	18. GetUpcomingCourses-hascontext
19. GetUpcomingCourses-newcourse	20. IntroExec

For each of the intents, a set of training phrases were input to enable the chat bot to recognize the intents from utterances.

Fulfillment: This is the code. This part of the conversation lets you pass on the request from your bot to an external source and get response and pass it back to the user. This is achieved via a webhook. Setting up a webhook allows you to pass information from a matched intent into a web service and get a result from it. Our webhook consists of Python scripts utilising fuzzy text libraries to match user-entered utterances to courses, disciplines and tracks recorded in the knowledge base. The rich message responses are also coded into the Python scripts.

Below are the key steps taken in developing the front-end in DialogFlow.

Create an intent

An intent map what a user says with what your agent does. To create an intent:

1. Click on the plus icon add next to Intents. You will notice some default intents are already in your agent. Just leave them be for now.

2. Enter a name for your intent. This can be whatever you'd like, but it should be intuitive for what the intent is going to accomplish.
3. In the Training Phrases section, enter examples of what you might expect a user to ask for. Since you're creating a Smart Academic Network Assistant (SANA) the agent, you want to include questions about the disciplines and courses available. The more examples you provide, the more ways a user can ask a question and the agent will understand. Enter these examples:
    - What are the courses
    - What are the areas of study?
    - What can I learn
    - What does ISS offer
    - What fields of study
    - What can I learn at ISS

In other intents, the training phrases may contain words that are colored and tagged as @sys.any entities or other customized entities. Such highlighted entities will facilitate the retrieval of the relevant information scraped from the ISS website which is stored in a pickle file and loaded in the app.py python script.

Add response

Now you'll add basic responses to the intent so the agent doesn't just sit there in awkward silence. Most of the created intent in this project will return dynamic responses. It uses fulfillment to return these dynamic responses. They consist of code that is deployed as a webhook and responds to HTTP requests from Dialogflow. Your fulfillment code processes the information from the matched intent and constructs a response to return to the user. More specifically, once user input is parsed, Dialogflow sends the name of the intent that was matched, the values of extracted parameters, and other metadata as a JSON payload in an HTTP POST request to your webhook. Your webhook constructs a response and sends it as a JSON payload to your Dialogflow agent, which then delivers the response to the user.

3.2 TECHNICAL SOLUTIONS

3.2.1 DATA SCRAPING AND COLLECTION

The data source for the information retrieval is the ISS-NUS website. We have two options to retrive the data:-

Retrive dynamically from the website at the point of time

Extract data before-hand and use as a repository for web-hook to retrive from

This agent uses the second option to improve system reliability. If we run the information retrieval in real-time, the agent is exposed to any potential website issues and may return errors. Moreover, dialoglow has a timeout of 5 seconds, and dynamic retrieval could potentially eat into that time and increase probability of timeouts.

To mitigate the risk, in the have chosen to run a scraper to extract the data before hand and this can be run regularly once a week to update the database.

For web-scraping, we considered Scrapy, Beautiful Soup and Selenium.

Scrapy, overall, is a web crawling framework written in Python. One of its main advantages is that it's built on top of Twisted, an asynchronous networking framework, which in other words means that it's: a) really efficient, and b) Scrapy is an asynchronous framework.

BeautifulSoup is used for extracting data points from the pages that are loaded. Beautiful Soup is quite robust and it handles nicely malformed markup. So, in other words, if we have a page that is not getting validated as a proper HTML but we know for a fact that it's a page and that it's an HTML specifically page, then we should give it a try scraping data from it with Beautiful Soup.

If the data is included in html source code, both frameworks can work fine and we can choose one we like. But in some cases the data show up after many ajax/pjax requests, the workflow make it hard to use Scrapy to extract the data. Scrapy only visit the url you told it, but Selenium will control the browser to visit all js file, css file and img file to render the page, that is why Selenium is much slower than Scrapy when crawling.

If the data size is big, Scrapy is the better option, but in this case our agent is designed only for an initial set of 140 courses.

Selenium is capable of handling the cases where the content being crawled is being added to the page via JavaScript, rather than baked into the HTML. JavaScript served with the page has the content already baked into it. The JavaScript is just there to do the templating or other DOM manipulation that puts the content into the page. The JavaScript is hitting a web API to load content.

Therefore looking at the above considerations, the selenium tool was used for webscraping.

Selenium, on the other hand, uses a driver that basically opens up a version of the web browser that can be controlled by python. This has the advantage that the website views you basically like any other human surfer allowing us to access information in the same way. It uses a web-driver package that can take control of the browser and mimic user-oriented actions to trigger desired events. This process is suitable for static content which is available by making an HTTP request to get the webpage content, but dynamic websites load the data from a data source (database, file etc) or require a few additional action events on the web page to load the data.

During the scraping process, any user action on a browser window can interrupt the flow and can cause an unexpected behavior. So, for scraping applications, it is crucial to avoid any external dependency while creating applications, such as browser. Headless browsers can work without displaying any graphical UI which allows applications to be a single source of interaction for users and provides a smooth user experience.

The knowledge base is extracted in the following structure:-

discipline kb			
<discipline-name>: dict			
	"name": str		
	"url": str		
	"description": str		
	"tracks": set		
		<track-name>: str	
	"courses": set		
		<course-name>: str	
track kb			
<track-name>: dict			
	"discipline": str		
	"courses": set		
		<course-name>: str	
	"levels": list		
		<level-name>: str	
	<level-name>: set		
		<course-name>: str	
course kb			
<course-name>: dict			
	"name": str		
	"url": str		
	"tags": set		
		<discipline-or-track-name>: str	
	"roadmap_tuples": set		
		(<discipline-name>, <track-name>, <difficulty-level>): tuple	
	"duration": str		
	"time": str		
	"enquiry-name": str		
	"enquiry-email": str		
	"enquiry-num": str		
	"intro": str		
	"classes": list		
		<class>: str	
	"takeaways": list		
		<takeaway>: str	
	"target": list		
		<target-audience>: str	
	"curriculum": list		
		<item>: str	
	"fees": dict		
		<sponsor-type>: dict	
			<applicant-type>: str

	"instructors": list		
		(<name>, <title>): tuple	
	"assessment": dict		
		"name": str	
		"items": list	
			<assessment-item>: str
	"soa": list		
		<soa-item>: str	
prep_info			
"t_and_c": str			
"commitment": str			
"wifi": str			
"venue": str			
"confirmation": str			
"general_enquiry": str			
funding_info			
"schemes": list			
	<scheme-name>: str		
<scheme-name>: list			
	<scheme-detail>: str		

3.2.2 USING FUZZYWUZZY FOR USER INPUTS

There is a possibility that the user may not know the exact name of the courses offered by ISS. So the user may end up typing “machine technology”, instead of “ machine reasoning” Moreover the user may also make a spelling mistake and type “ machnie” instead of “machine”. Therefore, instead of searching the user input to match the course name, we have used the fuzzywuzzy package.

FuzzyWuzzy has a ratio function that computes the standard Levenshtein distance similarity ratio between two sequences of characters. The Levenshtein distance is a metric to measure how apart are two sequences of words. In other words, it measures the minimum number of edits that one needs to do to change a one-word sequence into the other. These edits can be insertions, deletions or substitutions.

The formal definition of the Levenshtein distance between two strings aa and bb can be seen as follows:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Where 1(ai≠bj)1(ai≠bj) denotes 0 when a=ba=b and 1 otherwise. It is important to note that the rows on the minimum above correspond to a deletion, an insertion, and a substitution in that order.

It is also possible to calculate the Levenshtein similarity ratio based on the Levenshtein distance. This can be done using the following formula:

(|a|+|b|)−leva,b(i,j)|a|+|b|(|a|+|b|)−leva,b(i,j)|a|+|b|

where |a||a| and |b||b| are the lengths of sequence aa and sequence bb respectively.

If we apply this function to the earlier example where we are trying to compare "Machine technology" to "Machine reasoning" we will see that these two strings are very likely the same since the Levenshtein distance is very small.

The extract function in fuzzywuzzy process finds the best matches in a list or dictionary of choices, return a list of tuples containing the match and it's score. If a dictionary is used, also returns the key for each match. This also takes care of the individual string match to take care of the selling mismatch.

Therefore, depending on the user inputs, we match it to the list of the courses and the disciplines and the package returns the course or the discipline that has the best matching ratio.

```
def match_disc(query: str) -> str:
    result = process.extract(query, kb.disc_kb.keys())

    return result[0]



def match_course(query: str) -> str:
    result = process.extract(query, kb.course_kb.keys())

    return result[0]
```

```
def match_any(query: str) -> str:
    disc_result = match_disc(query)
    course_result = match_course(query)

    if course_result[1] > course_result[1]:
        return "course", course_result[0]
    else:
        return "discipline", disc_result[0]
```

4 SYSTEM ARCHITECTURE

4.1 PROJECT SCOPE

The chat bot is implemented using DialogFlow platform together with python script to provide dynamic response through the fulfillment. *Figure 3*, the system architecture diagram, illustrates how the application in the front-end app or devices to interact with the user and transmit the utterance to DialogFlow to determine the correct intent, and subsequently return the corresponding response through the fulfillment component.

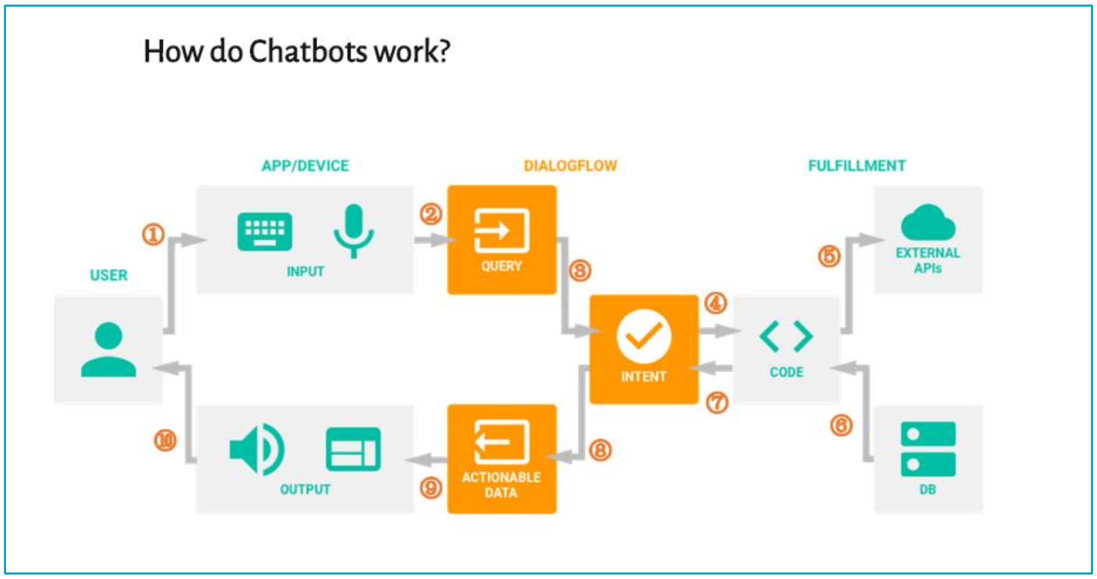


Figure 3: Typical system architecture of the deployed DialogFlow chatbot

There are detailed steps:

1. A user sends a text/voice message to a device or an App
2. The App/Device transfers the message to Dialogflow
3. The message is categorized and matched to a corresponding intent (Intents are defined manually by developers in Dialogflow)
4. We define following actions for each intent in the fulfillment (Webhook).
5. When a certain intent is found by Dialogflow, the webhook will use external APIs to find a response in external data bases.
6. The external data bases send back required information to the webhook.
7. Webhook sends formatted response to the intent.

8. Intent generates actionable data according to different channels.
9. The actionable data go to output Apps/Devices.
10. The user gets a text/image/voice response.

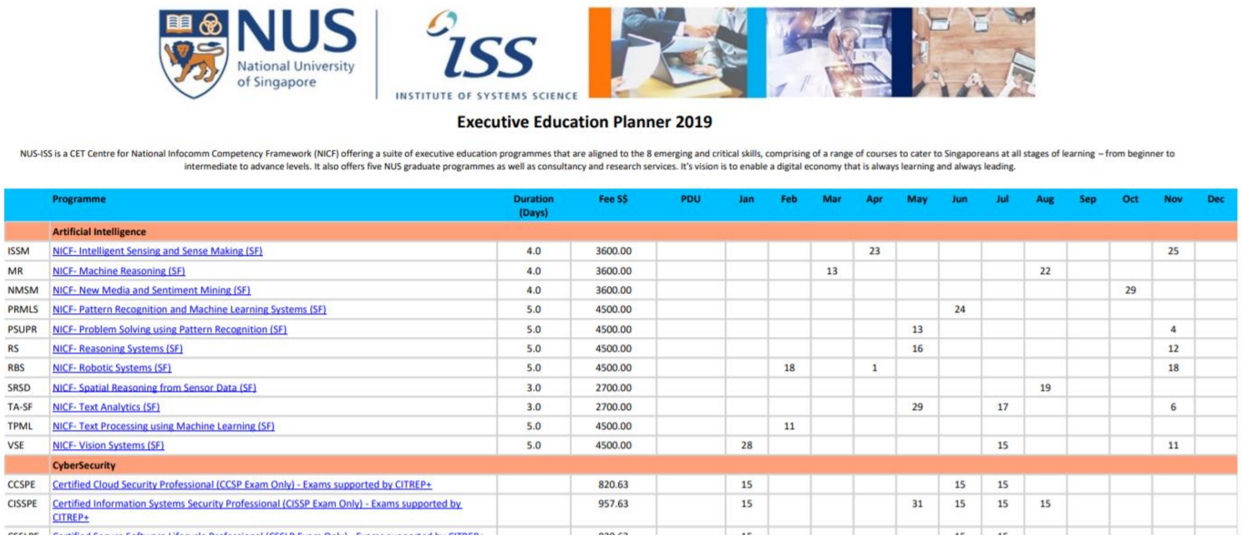
Its scope is limited by the data scraped from the ISS website and compiled

4.2 ASSUMPTIONS

The following data were used for the development of this ISS chat bot:

(i) 9 Disciplines from the Executive Programme

(ii) 100 + courses



NUS National University of Singapore | ISS INSTITUTE OF SYSTEMS SCIENCE

Executive Education Planner 2019

NUS-ISS is a CET Centre for National Infocomm Competency Framework (NICF) offering a suite of executive education programmes that are aligned to the 8 emerging and critical skills, comprising of a range of courses to cater to Singaporeans at all stages of learning – from beginner to intermediate to advance levels. It also offers five NUS graduate programmes as well as consultancy and research services. It's vision is to enable a digital economy that is always learning and always leading.

	Programme	Duration (Days)	Fee S$	PDU	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
	Artificial Intelligence															
ISSM	NICF- Intelligent Sensing and Sense Making (SF)	4.0	3600.00					23							25	
MR	NICF- Machine Reasoning (SF)	4.0	3600.00				13					22				
NMSM	NICF- New Media and Sentiment Mining (SF)	4.0	3600.00											29		
PRMLS	NICF- Pattern Recognition and Machine Learning Systems (SF)	5.0	4500.00							24						
PSUPR	NICF- Problem Solving using Pattern Recognition (SF)	5.0	4500.00						13						4	
RS	NICF- Reasoning Systems (SF)	5.0	4500.00						16						12	
RBS	NICF- Robotic Systems (SF)	5.0	4500.00			18		1							18	
SRSD	NICF- Spatial Reasoning from Sensor Data (SF)	3.0	2700.00									19				
TA-SF	NICF- Text Analytics (SF)	3.0	2700.00						29		17				6	
TPML	NICF- Text Processing using Machine Learning (SF)	5.0	4500.00			11										
VSE	NICF- Vision Systems (SF)	5.0	4500.00		28						15				11	
	CyberSecurity															
CCSPE	Certified Cloud Security Professional (CCSP Exam Only) - Exams supported by CITREP+		820.63		15					15	15					
CISSPE	Certified Information Systems Security Professional (CISSP Exam Only) - Exams supported by CITREP+		957.63		15				31	15	15	15				

(iii) 34 paths pertaining to the 9 Disciplines.

The above resources are deemed to be current based on the team's collation efforts in May 2019.

4.3 SYSTEM'S FEATURES

As the chat bot uses some form of rich responses, this POC chat bot is to be tested using Google Assistant. This platform provides us with the following interactive features:

- Voice input interaction
- Text-to-voice response
- Interactive lists and buttons
- Compatibility with mobile devices

4.4 LIMITATIONS

For a chatbot to be effective at processing natural language input from the user, it is necessary for it to be trained on a large dataset of utterances. In this proof of concept, our chatbot lacks such a dataset, and is restricted to a limited number of training phrases arbitrarily identified by our team members. This means that the chatbot is potentially unable to pick up the correct user intents if the input utterance differs from our limited variety of training phrases.

5 CONCLUSION & REFERENCES

5.1 IMPROVEMENTS AND FUTURE ENHANCEMENTS

Given that the application is a proof of concept project, the team had brainstormed and are looking forward to incorporating the following enhancements subject to availability of resources:

1) Refinement of the entities and intent utterances such that it minimize conflicts
2) More rich message responses
3) Extent the chat bot answering capabilities to cover Stackable Programme and Graduate Programme related queries.
4) Incorporate RASA into webhook to provide better Natural Language Understanding (NLU) for customized situations where current NLU capabilities in DiaglogFlow does not cater.
5) Use of a dataset of training phrases – possibly collected from NUS customer service records or other similar databases – for future training of the agent.
6) Hosting the webhook on an online cloud platform such as Heroku.
7) Periodic automated scraping of the NUS-ISS website to maintain an up-to-date knowledge base.

6 BIBLIOGRAPHY

1) Brandtzaeg, Petter & Følstad, Asbjørn. (2017). Why people use chatbots. In Proceedings of the 4th International Conference on Internet Science (INSCI '17). Spring, Cham, 377-392. DOI:https://doi.org/10.1007/978-3-319-70284-1

2) A brief introduction to Chatbots with Dialogflow
https://www.margo-group.com/en/news/a-brief-introduction-to-chatbots-with-dialogflow/

3) DialogFlow, Overview
https://dialogflow.com/

4) How to build a chatbot with Dialog flow | Chapter 1—Introduction
https://medium.com/swlh/how-to-build-a-chatbot-with-dialog-flow-chapter-1-introduction-ab880c3428b5

5) Dialogflow - Wikipedia
https://en.wikipedia.org/wiki/Dialogflow