

A Chatbot Framework for Yioop

A Project Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements of the Degree

Master of Science

By

Harika Nukala

May 2017

© 2017

Harika Nukala

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Master's Project Titled

A Chatbot Framework for Yioop

by

Harika Nukala

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2017

Dr. Chris Pollett	Department of Computer Science
-------------------	--------------------------------

Dr. Robert Chun	Department of Computer Science
-----------------	--------------------------------

Dr. Leonard Wesley	Department of Computer Science
--------------------	--------------------------------

## ABSTRACT

### A CHATBOT FRAMEWORK FOR YIOOP

by Harika Nukala

Over the past few years, messaging applications have become more popular than Social networking sites. Instead of using a specific application or website to access some service, chatbots are created on messaging platforms to allow users to interact with companies' products and also give assistance as needed. In this project, we designed and implemented a chatbot Framework for Yioop. The goal of the Chatbot Framework for Yioop project is to provide a platform for developers in Yioop to build and deploy chatbot applications. A chatbot is a web service that can converse with users using artificial intelligence in messaging platforms. Chatbots feel more like a human and it changes the interaction between people and computers. The Chatbot Framework enables developers to create chatbots and allows users to connect with them in the user chosen Yioop discussion channel. A developer can incorporate language skills within a chatbot by creating a knowledge base so that the chatbot understands user messages and reacts to them like a human. A knowledge base is created by using a language understanding web interface in Yioop.

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my project advisor, Dr. Chris Pollett, for his guidance, motivation and immense knowledge during this project. Without his precious support, this project would have not been possible. I want to take this opportunity and extend my appreciation to everyone who helped me in this project secondarily.

I sincerely thank my committee members, Dr. Robert Chun and Dr. Leonard Wesley, for their valuable suggestions and encouragement. Thank you all for the support.

## Table of Contents

<b>INTRODUCTION .....</b>	<b>9</b>
<b>BACKGROUND .....</b>	<b>11</b>
2.1 Facebook bot framework .....	11
2.2 Microsoft bot framework.....	13
<b>DESIGN AND ARCHITECTURE.....</b>	<b>15</b>
<b>IMPLEMENTATION .....</b>	<b>20</b>
4.1 Chatbot API .....	20
4.2 Language Understanding API.....	22
4.2.1 Plan bot story.....	23
4.2.2 Create a Bot Story .....	23
4.2.3 Add Intents .....	23
4.2.4 Add Entities.....	25
4.2.5 Add Expressions.....	25
4.3 How the Language understanding API works.....	28
4.3.1 Classifying intent .....	29
4.3.2 Extracting Entities .....	33
<b>TESTING.....</b>	<b>36</b>
<b>CONCLUSION AND FUTURE WORK .....</b>	<b>41</b>
<b>REFERENCES.....</b>	<b>43</b>

## List of Figures

Fig. 1. Data Flow of Facebook Bot Framework .....	12
Fig. 2. Control Flow of Microsoft Bot Framework .....	14
Fig. 3. Yioop's Chatbot Framework Design .....	15
Fig. 4. Architecture of Yioop's Chatbot Framework.....	16
Fig. 5. Control Flow between two main components of Yioop's chatbot Framework.....	17
Fig. 6. Schema of a Bot Knowledge Base .....	18
Fig. 7: Sample Knowledge base.....	19
Fig. 8. Web interface to create a bot user account .....	21
Fig. 9. Authentication flow of a bot web service.....	22
Fig. 10. Web interface to create a Bot story and add an Intent.....	24
Fig. 11. Web interface to add an Entity .....	25
Fig. 12. Web interface to edit an entity and add new entity values .....	26
Fig. 13. Web interface to add an Expression and label it .....	27
Fig. 14: Execution flow of language processing.....	29
Fig. 15. Function to classify intent.....	30
Fig. 16. Function to extract entities .....	34
Fig. 17. Knowledge base of weather bot for experiment 1 .....	37
Fig. 18. Conversation between a user (John) and a weather bot (wbot) in experiment 1 .....	38
Fig. 19. Entity knowledge base of weather bot after experiment 1 .....	38
Fig. 20. Conversation between a user (John) and a weather bot (wbot) in experiment 2 .....	39
Fig. 21. Expression Knowledge base of weather bot after experiment 2 .....	40
Fig. 22. Entity knowledge base after experiment 2 .....	40

## **List of Tables**

Table 1. Expression.....	32
Table 2. Term Frequency .....	32



# CHAPTER 1

## INTRODUCTION

A chatbot is a computer program that can converse with humans using artificial intelligence in messaging platforms. The goal of the project is to design and develop a chatbot framework which provides a facility for developers to create chatbots for Yioop. Yioop is an open source, PHP search engine that can be configured to allow users to create discussion groups, blogs, wikis..., and so forth. Yioop provides all the basic features of web search portal. It has its own account management system with the ability to set up groups that have discussions boards. Groups are collections of users that have access to a group feed. The user who creates a group is set as the initial group owner. Posts are grouped by thread in a group containing the most recent activity at the top [6].

Chatbots are created using two powerful APIs in Yioop. One is the chatbot API, that acts as a connector and routes messages between users and chatbot. By default, a chatbot is simple. A developer needs to program a bot service to define their behavior and make them intelligent. An artificially intelligent bot can understand the meaning of a user request and converse on messaging platforms. We offer an API called Language understanding API that makes bots more like a human by providing functionalities that recognize a user's intent by using machine learning concepts and allow bot to perform required action. The entities are key information to perform an action. These entities are extracted from user's message by using matching keywords or the similar wording pattern, from an existing database.

A developer must have expertise and special knowledge in a few areas like machine learning and artificial intelligence to create a chatbot and connect it with users in a

conversational interface. To make it easier, companies like Facebook and Microsoft introduced bot frameworks.

The rest of the report is organized as follows. Chapter 2 provides background information on Facebook bot framework and Microsoft bot framework. Chapter 3 outlines the architecture we designed to build the chatbot Framework for Yioop. Chapter 4 describes high level implementation of Yioop Chatbot Framework and outlines approaches and algorithms used to build this system. Chapter 5 discusses integration of required components by providing an example chatbot created on Yioop called Weather bot and experimental results. Chapter 6 concludes the topic.

## CHAPTER 2

### BACKGROUND

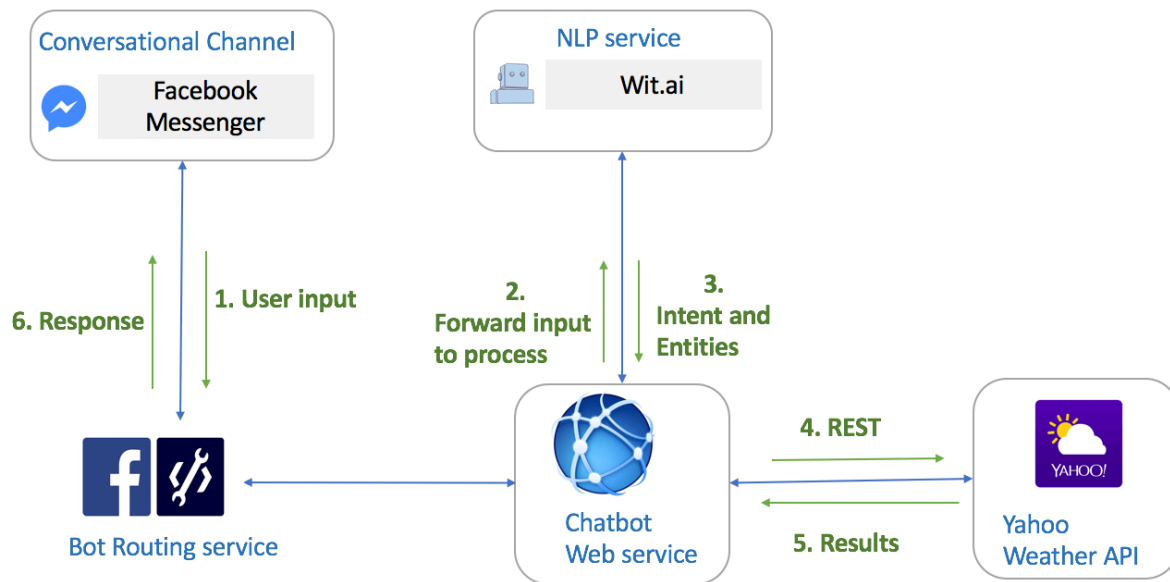
In this chapter, we discuss the famous bot frameworks which are available to public to get an idea of a chatbot framework. The bot frameworks are developed to solve the problems met by the developers while writing bots. Those are: Bots require language understanding skills; and they must connect to the users in a conversation interface, the user chooses. It is difficult to deal with these requirements by developers. Bot framework provides tools and services which makes a developer's job easier.

Facebook and Microsoft have large-scale bot frameworks designed and developed to produce a mass number of chatbots. These frameworks provide tools that help developers to build bots and define their behavior using a programming language.

#### **2.1 Facebook bot framework**

Billions of users use Facebook Messenger, so Facebook bot uses Messenger as the messaging platform. Facebook Messenger platform provides Receive and Send API that allows developers to create bots in order to interact with businesses [2]. These APIs not only support text conversations, but also multimedia conversations like sending images, gifs, videos, and so forth. To enable developers to create more complex bots and make use of machine learning which helps bot understand intent of a user's message, Wit.ai, a Facebook's Bot Engine can be incorporated [1]. This bot engine relies on the idea of machine learning, therefore, makes Facebook bots more powerful. Sample conversations must be provided to train Wit.ai's intent so that the bot can handle many different variations of a sentence. These sample conversations are called Stories, and include bot actions. Bot Engine builds a machine learning model that works

with stories in the dataset. As the dataset grows, the model becomes better. Bot Engine predicts the next action of a bot at every conversation. The prediction depends on the stories, the bot has trained. The definition of the action must be in the application code. The code can be written in any programming language and call any external API as you need.



**Fig. 1. Data Flow of Facebook Bot Framework**

The Facebook bot interaction with other components is shown in Figure 1 and explained below. A user sends a message “what is weather in Seattle” to Chatbot service through Facebook Messenger (a Chat Client). The Chatbot service posts the message to Wit.ai (a Machine Learning, Natural Language Processing Engine). Wit.ai extracts user’s intent “getWeather” and entities “Seattle” from the message and sends them back to Chatbot service. An Intent is used to call upon external data service if required to find desired data. Chatbot service builds data into a proper response “The weather is 49 degrees and rain in Seattle” and sends to Facebook Messenger for display.

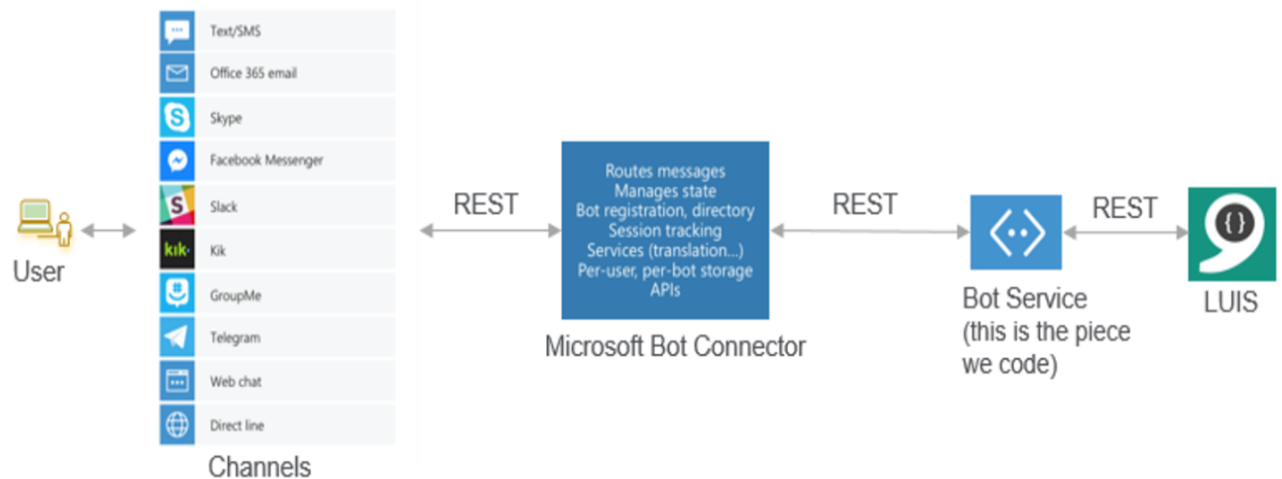
## **2.2 Microsoft bot framework**

Microsoft announced its Bot Framework roughly at the same time as Facebook. As with the Facebook's offering, Microsoft provides an SDK that can be viewed as two components:

1. Bot Connector, the integration component
2. LUIS.ai, the natural language understanding service

These two components are independent to each other. Bot Connector allows bots to send and receive messages to messaging platforms. This component helps to associate with various conversational channels like Facebook Messenger, Slack, Skype, Email, GroupMe and SMS [3]. Microsoft Bot Framework is a comprehensive offering to build and deploy chatbots for users to enjoy conversation experiences.

The diagram below shows how each component interacts. A user chooses a channel to send a message to a bot. The message is directed through the Microsoft Bot Connector, which sends a POST request to a bot web Service. The request sent by a bot connector to a bot service has a user's message. Once the bot service receives the request, it performs necessary action and reply back to the user. A bot is not intelligent itself. The intelligence comes when we incorporate Language Understanding Intelligent Service LUIS, Natural Language Processing as a Service is one of the Cognitive Services provided by Microsoft. When a bot is integrated with LUIS, the message received from the user is sent to LUIS to understand what user's intention and then reply back accordingly.



***Fig. 2. Control Flow of Microsoft Bot Framework***

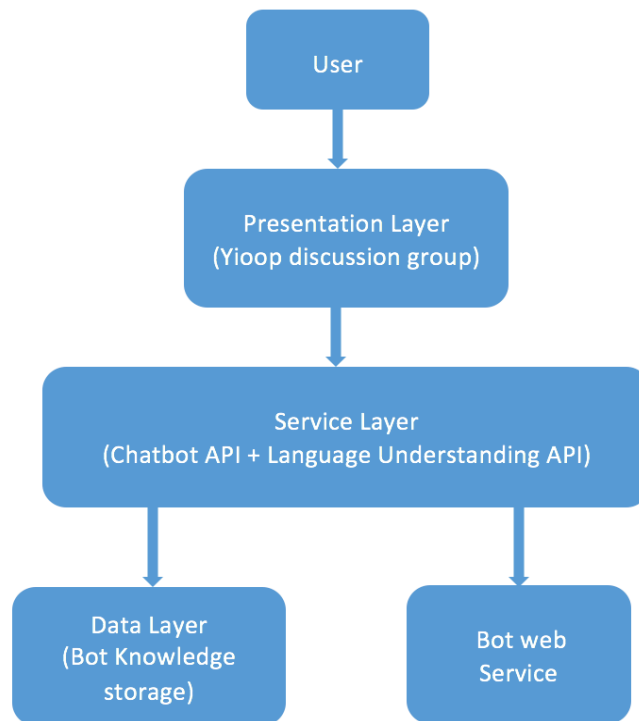
We have discussed two different bot frameworks. Both of these bot frameworks provide tools to train and build bots. Chatbots created using Microsoft Bot Framework can be published over various communication platforms as compared to the chatbots created using Facebook Bot Framework, only work on the Facebook Messenger platform. The Facebook bot Framework uses Wit.ai and the Microsoft bot Framework uses Luis.ai as a natural language processing engines respectively. Both wit.ai and luis.ai are external web services which are consumed as a backend service in chatbot applications using a REST call. This makes interactions between users and bots slower.

The aim of our project is to create a Chatbot Framework for Yioop, with a similar kind of end goal in mind. It is advisable to minimize the external API calls; therefore, we created our own Language processing API in Yioop to make conversations faster between users and bots, which we will discuss further. In the next chapter, we will discuss about the design and architecture of this project.

## CHAPTER 3

### DESIGN AND ARCHITECTURE

As we have already discussed, Bots require keywords for processing and taking action based on given keywords or conversation context. They must also have language skills for communication, and they should also interact with users over some channel. The Yioop's chatbot framework provides chatbot API and language understanding API to solve these problems.

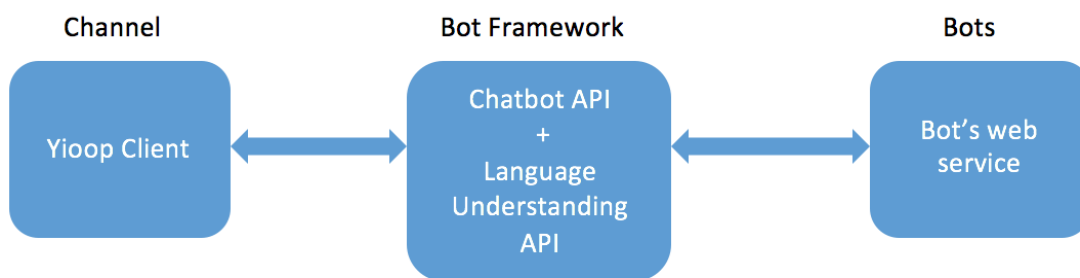


***Fig. 3. Yioop's Chatbot Framework Design***

Figure 3 shows design of the Yioop's Chatbot Framework we implemented. The user requests or chats with bot in a message interface. Like Facebook, Yioop provides a conversational interface to connect users with bots called Yioop discussion group, which is the

presentation layer. The two APIs that are responsible to send and receive messages, process user's request and apply language processing skills are in the service layer. The bot web service handles the POST requests and performs the respective action. Data layer has the bot knowledge base (See Figure 7).

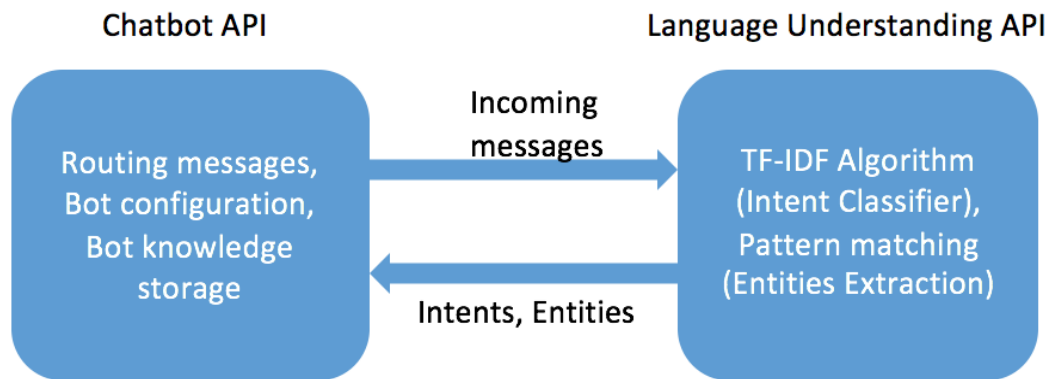
The integration and control flow between the components of Yioop is shown in the Figure 4 and explained below.



***Fig. 4. Architecture of Yioop's Chatbot Framework***

The User converses with a chatbot using Yioop discussion group as a messaging platform. The user's message is routed through the chatbot framework. The Framework contains two APIs which receives the user's message, requested user details and bot details. These APIs process the message and invokes the bot by sending a POST request to the bot web service with the processed message. The Control flow between these two APIs is shown in the Figure 5.

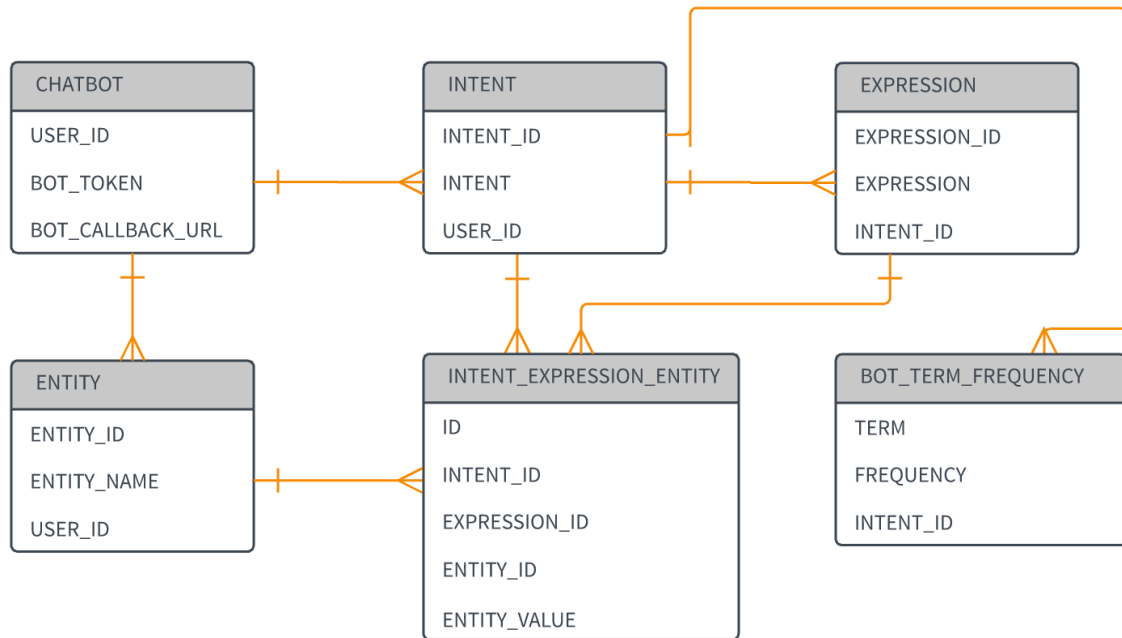




***Fig. 5. Control Flow between two main components of Yioop's chatbot Framework***

The Chatbot API allows developers to create chatbots and configure them in Yioop groups to send and receive messages to thread. Users can interact with a chatbot from any group thread that a bot is configured to work for. A developer can create a knowledge base using language understanding web interface to make bots understand and react to natural language interactions more like a human. Language Understanding API helps bot to understand the user's intent and extract knowledge from the messages which helps to improve bot's capability.

The Schema of a Bot Knowledge storage is shown in the Figure 6. The Bot Knowledge Storage contains intents, entities and the respective set of expressions which are specific to the bot domain/story. This knowledge base is like a brain for a bot which has the knowledge what bot already learnt or trained from developers and it helps the bot to deal with new conversations. The new conversations are continuously stored in the knowledge base because they are the new learnings for bot.



**Fig. 6. Schema of a Bot Knowledge Base**

#### INTENT

TABLE	INTENT	Search	Show All	Add	Duplicate	Edit	Del
INTENT_ID	INTENT	USER_ID					
1	getWeather	3					
2	getStockPrice	3					

#### ENTITY

Structure

Browse & Search

Execute SQL

DB Settings

TABLE

ENTITY

Search

Show All

Add

Duplicate

Edit

Del

ENTITY_ID	ENTITY_NAME	USER_ID
1	location	3
2	company	3

## EXPRESSION

TABLE	EXPRESSION	Search	Show All	Add	Duplicate	Edit	Del
EXPRESSION_ID	EXPRESSION	INTENT_ID					
4	what is the weather in &location	1					
5	get me weather of germany	1					
6	get me weather of Delhi	1					
7	what is stock price of &company	2					
8	get me weather of seattle	1					
9	get me stock price of apple	2					

*Fig. 7. Sample Knowledge base*

The functionalities of the two APIs and use of the knowledge base are explained in the next chapter in more detail.

## CHAPTER 4

### IMPLEMENTATION


In this chapter, we discuss the high-level implementation of the two APIs of the Yioop Chatbot Framework and the integration between them.

#### **4.1 Chatbot API**

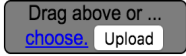
The chatbot API is a service that allows bot users connect easily with the users on a group discussion only when the bot users already joined the group. This service receives messages from the thread and passes them to chatbot application and then sends responses back to the thread. This service calls Language understanding API to generalize the user request so that bot understand it.

To configure a bot, as a first step, the administrator must enable bot users feature in server settings of Yioop. The next step is to create accounts for bots. The procedure of creating a bot user is similar to creating a normal user. Bot users have many of the similar features as the normal users in Yioop. They have profile photos, names, and passwords, they can be specified in a post, they can reply to the post messages, and they can be invited to and removed out of groups. The main difference is that the bot users are controlled programmatically by a bot unique token and bot callback URL. The Callback URL represents the chatbot application endpoint. When the user sends a message, the chatbot API posts the request to Callback URL, so a bot Callback URL should be specified when you create a bot user (see Figure 7).

## Account Details



Username:   
 First Name:   
 Last Name:   
 Email:   
 Advertise With Us ☐  
 Bot User ☒  
 Bot Unique Token   
 Bot Callback URL   
Password:

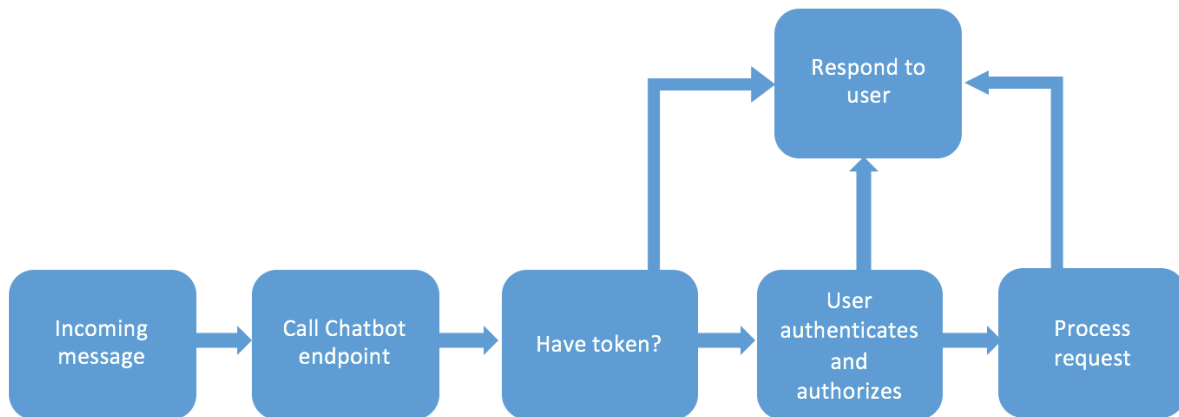


***Fig. 8. Web interface to create a bot user account***

Yioop supports group conversations with multiple users, including other bots. In the group thread, a user starts the conversation and bots simply responds to messages that the user sends. In order to converse with a bot in a group, the bot users must have joined the group. To converse with the bots, a user has to mention the bot name with '@' in the message (*@bot\_name*). The message is processed to get the values of the required properties like the username, bot callback URL, bot unique token, and conversation. If the requested bot already joined the group, the message is sent to bot web service (Bot Callback URL).

All of the requests sent to a bot's endpoint must include the user's message and a bot unique token. A cryptographic hash signature is used for the bot unique token and sends it to endpoint for authentication. To ensure that posts made to the bot's endpoint by chatbot API actually came to the requested bot, it is highly recommended that chatbot application verifies the authenticity of the requests. To verify the authenticity of the token sent by the chatbot API, the bot application has a procedure that extracts the token from the HTTP POST request, parses the token, verify its contents, and verify its signature (see Figure 8). The application checks whether

token has not expired and has a valid cryptographic signature. If the token does not meet these required conditions, bot terminates the request by returning “There was a configuration issue with the query” message.



*Fig. 9. Authentication flow of a bot web service*

## **4.2 Language Understanding API**

The main problem is the capability of the computer to understand what a user wants in human-computer conversations. Language Understanding API is designed and developed to allow developers of Yioop to create chatbot applications that can understand natural language and react to user requests accordingly. This API will take the sentences sent by user in a conversation and interpret intents (the intentions user convey) and extract entities (key information relevant to the intent) using knowledge base.

By using the Language Understanding web interface, a user can create a knowledge base for a chatbot with a set of intents, entities and the respective set of expressions that are relevant to chatbot story’s domain. For example, for a travel agent chatbot, a user might say “Book me a flight from San Jose to New York.” In this expression, the user’s intention is to “BookFlight”, “San Jose” and “New York” are the entities. An Intent is defined as the necessary action and

usually contain a verb, in this case “Book”. The entity is information relevant to the intent, in this case “San Jose” and “New York” are the source and destination location entities. Example expressions has to be provided for every intent and label the entities, which helps to predict intent for new expressions and operates the respective action.

The following sections discusses on how to create a knowledge base for a chatbot.

#### **4.2.1 Plan bot story**

All Chatbot stories are focused on a domain-specific subject, for example, booking of flights, get weather updates or get stock prices, etc. It is preferable to plan a bot story before creating it in Yioop.

Consider an example of a weather chatbot, user should determine the intents and entities that are significant to weather chatbot task. In a weather chatbot story, users would like to get an update of the weather at specific location. Thus, “getWeather” would be the relevant intent. For weather updates, some key information is needed such as the location, date and time and these are considered as entities. A user should create a bot story with an outline of intents and entities and define these intents and entities for example expressions in the interface.

#### **4.2.2 Create a Bot Story**

A user can create and manage a bot story in the respective bot user account. One can create bot story by clicking Bot Story on the left navigation bar of the Yioop web page. The Figure 9 shows an example weather bot story. Start the bot story by adding intents.

#### **4.2.3 Add Intents**

An Intent is the intention or necessary operation that is conveyed by the user through the sentences. Language understanding API service matches the user request with Intents of the bot

story to determine the action that should be performed by a chatbot. So, the intents must be added with example expressions to help chatbot understand user requests and respond to users properly.

The chatbot should be taught to identify user request that are relevant to the bot story. For example, a weather chatbot has learnt “What is weather in San Jose” with “getWeather” intent, if a user requests “Get me weather info of Seattle” to a weather chatbot, the bot should recognize the statement and matches to “getWeather” intent and perform necessary operation.

A user who operates the bot account can add and manage intents from the Intents tab in the bot story page. The following is the procedure for adding intent in a Weather bot story page. To add an intent:

1. Open a bot user account (e.g. Weather Bot) and click Bot story in the left panel of the web page, and then click Intents tab.
2. Type in the Intent box and click save.

One can edit or delete intents from the chatbot’s intents list (see Figure 9).

The screenshot shows the Voop! Admin [Bot Story] interface. On the left is a sidebar with three main sections: 'Account Access' (Manage Account), 'Social' (Manage Groups, Feeds and Wikis, Mix Crawls), and 'Chat Bot' (Bot Story). The 'Bot Story' option is selected. The main content area has tabs for 'Intent', 'Entity', and 'Expression', with 'Intent' being the active tab. Below the tabs is a form with an 'Intent' label, a text input field, and a 'save' button. Below the input field is a table with two columns: 'Name' and 'Actions'.

Name	Actions
Add	<a href="#">Edit</a> <a href="#">Delete</a>
getWeather	<a href="#">Edit</a> <a href="#">Delete</a>

**Fig. 10. Web interface to create a Bot story and add an Intent**



#### 4.2.4 Add Entities

Entities are the key information for a domain-specific chatbot. An entity is a collection of similar objects like a location, person's name, number. Entities are the key data relevant to an intent and they are necessary for a bot to perform a specific task. For example, in a weather bot, the entities may include "location", "date" and "time", which are key parameters to the "getWeather" intent. In a flight booking bot, the "location", "date", "airline", "travel class" and "tickets" are key information to the "BookFlight" intent. Therefore, these parameters are added as entities in a bot story (see Figure 10). A user should create entities that are relevant to the intent and required by a chatbot to perform a task.

To add an entity:

1. Login to the chatbot account and click Bot Story in the left panel.
2. Go to the Entity tab in the page and type in Entity box and then click Save.

The screenshot shows the Voop! Admin [Bot Story] interface. On the left is a sidebar with navigation links: Account Access (Manage Account), Social (Manage Groups, Feeds and Wikis, Mix Crawls), and Chat Bot (Bot Story). The main content area has three tabs: Intent, Entity (selected), and Expression. Below the tabs is a form with an 'Entity:' label and a text input field. A 'Save' button is to the right of the input field. Below the input field is a table with two columns: 'Entity Name' and 'Actions'.

Entity Name	Actions
location	<a href="#">Edit</a> <a href="#">Delete</a>
number1	<a href="#">Edit</a> <a href="#">Delete</a>

**Fig. 11. Web interface to add an Entity**

One can edit or delete entities from the Entity list of a chatbot story on the Entity tab.

Figure 11 shows the web interface to edit an entity and add new entity values for an entity. Entity values are like instances of a class. For example, *location* is an entity and *San Jose, Chicago*,

*Seattle* are the entity values of location entity. One can add any number of entity values for an entity.

The screenshot shows the Voop! Admin [Bot Story] interface. On the left is a sidebar with three main sections: 'Account Access' (Manage Account), 'Social' (Manage Groups, Feeds and Wikis, Mix Crawls), and 'Chat Bot' (Bot Story). The main content area has three tabs: 'Intent', 'Entity' (selected), and 'Expression'. The 'Edit Entity' form is displayed, showing the entity name 'location' and a list of entity values: 'chicago', 'india', 'newyork', and 'seattle'. Each value has a 'Delete' link next to it. There is a 'Save' button at the bottom of the form. Below the form is a table listing existing entities and their actions.

Entity Name	Actions
location	<a href="#">Edit</a> <a href="#">Delete</a>
number1	<a href="#">Edit</a> <a href="#">Delete</a>

*Fig. 12. Web interface to edit an entity and add new entity values*

#### **4.2.5 Add Expressions**

Expressions are example sentences of user requests that a chatbot is expected to grasp and understand. For each intent, there must be example expressions added in a bot story. The Language Understanding API uses these expressions to teach chatbot understand similar contexts. Adding more example expressions and labeling them improves the ability of a chatbot.

Including as many sentence variations as user tend to say or request to a specific bot, enhances the chatbot's language understanding experience. Also, adding more relevant example expressions to an intent for a bot story helps the language understanding API in predicting the relevant intent for a new expression. For example, the expression "what is weather in Seattle" may have variations such as "Get me weather forecast of Seattle", "Is it sunny or rainy in

Seattle”, “Tell me the weather in Seattle”, “How is weather in Seattle”, and “Is it raining in Seattle”.

Expressions are added by selecting the relevant intent on the Expression tab of the bot story page. The following steps describe how to add example expressions to an intent (e.g. “getWeather” intent for the Weather bot).

To add an expression:

1. Login to the chatbot account and click Bot Story in the left panel.
2. Go to the Expression tab in the page and Select **getWeather** from the Intent list.
3. Type the expression in Expression box.
4. Label **Seattle** as entity value to the entity name **location** (see Figure 12). If there is no entity name to label, then add entity name in the Entity tab (Check **Add Entities** section, to add an entity).

The screenshot shows the Voop! Admin [Bot Story] interface. On the left is a sidebar with navigation links: Account Access (Manage Account), Social (Manage Groups, Feeds and Wikis, Mix Crawls), and Chat Bot (Bot Story). The main content area has three tabs: Intent, Entity, and Expression. The Expression tab is active, showing the 'getWeather' intent selected in a dropdown. Below this, the 'Expression' field contains the text 'what is weather in Seattle'. Under the heading 'Bot User has following Entities:', there are two entity fields: 'location' with the value 'Seattle' and 'number1' which is empty. A 'Save' button is located below these fields. At the bottom, there is a table listing existing expressions.

Expression	Name	Actions
Tell me weather in new Jersey	getWeather	<a href="#">Edit</a> <a href="#">Delete</a>
how is weather in &location	getWeather	<a href="#">Edit</a> <a href="#">Delete</a>

*Fig. 13. Web interface to add an Expression and label it*

#### **4.2.5.1 Label Expressions**

Expressions are labeled in terms of intents and entities. It can be done before or after adding expressions. The web interface to add and label expression is shown in Figure 12.

##### **Intent label**

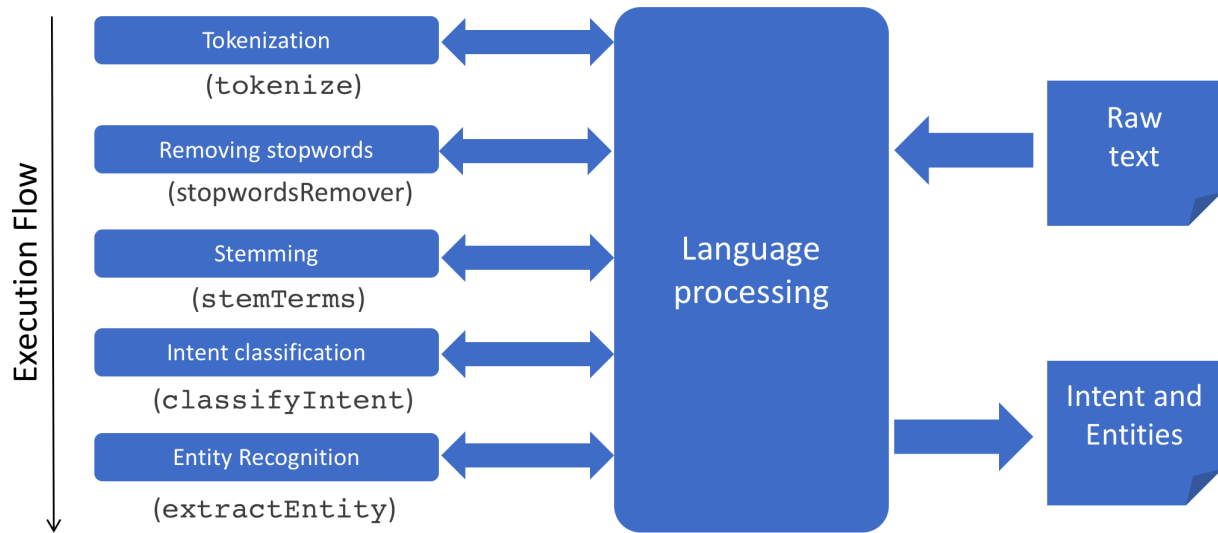
Selecting an intent while adding an expression means that it is labeled under this intent. That is how an expression gets an intent label. A user can change the intent label of one or more expressions. To do this, select the expression that needs to be edited, click edit expression, and then select a different intent from the list of intents.

##### **Entity label**

Entities need to be added in Entity tab in order to label them on an expression in a bot story. We map the word that needs to be labelled from an expression to an existing entity. For example, the expression “What is weather in Seattle” is mapped to the “getWeather” intent in Weather bot story, map “Seattle” to the “Location” entity. “Location” must be added as the entity in the Entity tab.

### **4.3 How the Language understanding API works**

There are two things that the Language understanding API can possibly do — Intent Classification and Entity Extraction. We will discuss them below.



*Fig. 14: Execution flow of language processing*

### **4.3.1 Classifying intent**

Given a sentence, this API classifies the sentence into one of the trained intents. The way we achieve this is by defining the intents and training with example sentences and manually classifying them which is explained above. This type of learning is called supervised learning and the algorithm used to classify is TF-IDF, term frequency-inverse document frequency. The function implemented to classify intent is shown in Figure 13 and discussed in detail below.

The first step is to break a user request into tokens to classify the intent in the Language Understanding API. For the most part, English tokens are delimited by whitespace. The goal of the tokenizer is to break the character sequence into a list of words. The example sentence “What is the weather in Seattle” of a weather bot is converted into tokens as shown below.

[What, is, the, weather, in, Seattle]

Note how most tokens are words you would find in the dictionary. The next step is to remove stopwords from the list of tokens which helps in generalizing the context. Stop words include

words which are prepositions, articles, pronouns and conjunction. These are usually most occurring words. The list becomes after removing stopwords [weather, Seattle].

```
function CLASSIFY-INTENT (message, bot) returns the top intent if exists,  
or null  
[intents]  $\leftarrow$  getIntent(bot)  
message  $\leftarrow$  removeStopwords(message)  
[terms]  $\leftarrow$  stemming(message)  
[intents with scores]  $\leftarrow$  TFIDF([terms], [intents])  
if intent scores > 0 store the message in the EXPRESSION table in database and  
return topIntent i.e., intent with highest score  
else return null
```

*Fig. 15. Function to classify intent*

#### **4.3.1.1 Stemming**

The next step is to apply stemming process to the list of terms that we have after removing stopwords. Stemming is the process of removing the ends and makes the set of words to its root word [7]. For example, the words like stems, stemming, stemmer and stemmed reduces to its root word stem. This process helps in generalizing the words in a sentence.

#### **4.3.1.2 TF-IDF**

The algorithm used to classify intent is described here. First, we assign a weight to each term in a set of sentences of an intent, which represents sum of number of occurrences of that term in each sentence 's'. We evaluate a score between a query term 't' and intent sentences 'S', based on the weight of 't' in 'S'. This procedure is denoted as term frequency  $tf_{t,s}$  where

subscripts signifying term  $t$  and set of sentences  $S$ , which is equal to summation of  $tf_{t,s}$  where subscripts referring term  $t$  and a sentence  $s$  all together [6].

$$tf_{t,S} = \sum_{s \in S} tf_{t,s}$$

The intent frequency  $df_t$  defined to be number of intents containing a term  $t$  in chatbot story. The total number of intents per chatbot is denoted as  $N$ , we define the inverse document frequency  $idf$  of a term  $t$  as

$$idf_t = \log \left( \frac{N}{1 + df_t} \right)$$

In simple words,  $idf_t = \log \left( \frac{\text{total number of intents}}{1 + \text{total number of intents having term } t} \right)$

To get a composite weight for each term in set of sentences of each intent, we combine term frequency and inverse document frequency. The  $tf-idf$  is defined as the product of term frequency and inverse document frequency.

$$tf - idf_{t,S} = tf_{t,S} \times idf_t$$

$tf-idf_{t,S}$  assigns a weight to term  $t$  in all the sentences of an intent. The weight is highest when term  $t$  occurs many times within a small set of sentences and lower when term  $t$  occurs few times in a set of sentences [6]. The idea is each intent is viewed as a vector with one component having each term with a corresponding weight that is given by  $tf-idf$ . For query terms that do not occur in any set of sentences, the weight is given 0.

The score of each intent is the sum of  $tf-idf$  weight of each query term in set of sentences.

$$Score(q, S) = \sum_{t \in q} tf - idf_{t,S}$$

The scores of each intent is calculated and returns the intent which has highest score. The TF-IDF model is explained with an example below.

Suppose the tables shown below are knowledge storage tables for a chatbot, the user requesting. **EXPRESSION** table having intents and respective Expressions and **TERM FREQUENCY** table having terms per intent and their frequencies which are calculated by using term frequency formula  $tf_{t,s}$

**Table 1. Expression**

Intent	Expression
getWeather	What is weather in <i>&amp;location</i>
getWeather	How is weather in <i>&amp;location</i>
getWeather	Is it rainy in <i>&amp;location</i>
BookFlight	Book flight from <i>&amp;source</i> to <i>&amp;destination</i>

**Table 2. Term Frequency**

Intent	Term	Frequency
getWeather	weather	2
getWeather	rainy	1
BookFlight	Book	1
BookFlight	flight	1

When a user requests “Tell me the weather in Seattle”, the tf-idf value is calculated for all the intents with respective to chatbot by considering each valuable term in query (stopwords are removed).

As we can see the Expression table has two unique intents, so value of N is 2 (N=2).

Terms of a query are [**weather, Seattle**] after removing stopwords.



For term **weather**, the total number of Intents with this term is one i.e., getWeather. Therefore,

$$\begin{aligned} \text{idf}(\text{weather}) &= 1 + \log\left(\frac{N}{1 + \text{total number of intents with term } \text{weather}}\right) \\ &= 1 + \log\left(\frac{2}{1+1}\right) = 1 \end{aligned}$$

$\text{tf}(\text{weather}, \text{getWeather}) = \text{frequency of term } \text{weather} \text{ in Term Frequency table} = 2$

$\text{tf}(\text{weather}, \text{getWeather}) = 0$

Now the tf-idf's of the term *weather* for each intent are calculated as:

$\text{tf-idf}(\text{weather}, \text{getWeather}) = \text{tf}(\text{weather}, \text{getWeather}) * \text{idf}(\text{weather}) = (2*1) = 2$

$\text{tf-idf}(\text{weather}, \text{BookFlight}) = \text{tf}(\text{weather}, \text{BookFlight}) * \text{idf}(\text{weather}) = 0$

tf-idf's of term *Seattle* for both intents are 0, as it never occurred in any expression.

Scores of each intent for query '*weather Seattle*' is calculated below:

$\text{Score}(\text{query}, \text{getWeather}) = \text{sum of tf-idf's of getWeather intent} = 2$

$\text{Score}(\text{query}, \text{BookFlight}) = \text{sum of tf-idf's of BookFlight intent} = 0$

Since the score of getWeather Intent is highest, getWeather is returned as intent for new query.

### **4.3.2 Extracting Entities**

The Language understanding API helps us by extracting words called entities from the given sentence. This API uses pattern matching algorithm to detect entities. This is achieved by training multiple sentences for each intent and labelling respective entities manually. The function implemented to extract entities is described in Figure 14.

#### **Pattern matching**

We use pattern matching and regular expressions heavily in our framework to extract the entities from the sentences. The entities which are special information in sentences are then used

to perform specific actions based on the use case. We rely on PHP language's regex capabilities to achieve this task. Regex is like a mini syntactical language which dictates how the overall structure of the sentence should be and then we could specify the relative position of entities inside those sentences.

```
function EXTRACT-ENTITIES (message, topIntent) returns list of entities  
if exists, or empty list  
[expressions] ← getExpressions(intent)  
FOR each expression in [expressions]  
  [entity_names] ← MATCH_REGEX (label_regex, expression)  
  expression ← REPLACE([entity_names], new_regex, expression)  
  [(entity-name, entity-value)] ← PATTERN-MATCH (expression, message)  
  if size of entity name, value pairs >0 store the entity values in  
  INTENT_EXPRESSION_ENTITY table in database and return list of entities i.e.,  
  [(entity-name, entity-value)]  
  else return empty list
```

***Fig. 16. Function to extract entities***

In this API, we provide a regular expression implementation that generalizes the sentences that are trained. The sequence of tokens that are labelled while adding an expression above are replaced with regular expressions in the form of groups.

This API matches the user's message with a similar wording pattern and apply pattern matching. Based on the sub-patterns which are specified in the form of groups, it can extract what characters/words have been matched out of the sentence. The extracted words are called entity values.

For example, the expression we trained is “what is weather in Seattle” and we labelled the “Seattle” to the “location” entity. So, the expression generalizes to “what is weather in &location”. When a user asks, “How is the weather in San Jose”, the classification algorithm gives the user’s intent as “getWeather” and then the API replaces labels in the sentences of the predicted intent in the knowledge storage to the regular expression in the form of groups. The trained sentence becomes “what is weather in (.\*)”. The pattern matching technique is applied on the trained sentence and the user’s message and extracts “San Jose” as an entity value for the “location” entity. These extracted entities are sent as a POST request to the bot web service to perform “getWeather” action. The result of this action is sent as a response to the user. The experimental results of these functionalities are shown in the testing section.

## CHAPTER 5

### TESTING

The main purpose of this project is to provide a chatbot framework for Yioop that helps developers to create and develop chatbots. After integrating the chatbot framework in the Yioop search engine, we performed some experiments to see how chatbot works and how they interact with users in Yioop discussion groups.

We created sample chatbot accounts for weather bot user as *wbot* and stock bot user as *StockBot* using bot configuration settings. To evaluate the chatbot framework, we need to create chatbot applications. We created a Weather Bot and Stock Bot services with unique bot token for authenticity. These bot service URL's and bot tokens must be provided as bot token and bot callback URL's in respective bot accounts in order to have a conversation with users in Yioop discussion groups. These bots must be added to the Yioop groups the user chosen to interact.

These chatbots are focused entirely on providing information and completing tasks for the humans they interact with. When a user requests a bot, the message is routed through chatbot API and language understanding API which recognizes the user's intent and extract entities by using existing knowledge base. This intent and entities are used to perform required action.

The Weather Bot service performs action that intend to give weather updates for user requested location. This bot service calls the yahoo weather service which is an external API to get weather information. The Stock Bot service calls the yahoo finance, an external API to get the current stock price of the user asked stock symbol.

Suppose the Weather Bot is trained with two sentences. The knowledge base of the weather bot is shown in the figure below. The bot has learnt two expressions with *getWeather* intent.

**yooop! - Admin [Bot Story]**

**Account Access**  
Manage Account

**Social**  
Manage Groups  
Feeds and Wikis  
Mix Crawls

**Chat Bot**  
Bot Story

**Intent** | **Entity** | **Expression**

**Intent** getWeather

**Expression**

**Bot User has following Entities:**  
location  
number1


Save

Expression	Name	Actions
how is weather in &location	getWeather	<a href="#">Edit</a> <a href="#">Delete</a>
what is the weather in &location	getWeather	<a href="#">Edit</a> <a href="#">Delete</a>


**Fig. 17. Knowledge base of weather bot for experiment 1**

In the Experiment 1, a user interacts with weather bot in a **weather updates** thread as shown in the figure below. Here, the user named *john* is requesting weather updates for a couple of locations by tagging weather bot as *@wbot* in the message and weather bot is responding with the weather information.

The bot recognized ‘Seattle’ and ‘San Jose’ as entity values for location entity and performed respective actions. Experiment 1 is passed as the bot responded with weather information for the requested location as shown in Figure 16. The Entity knowledge base is updated with new entity values ‘*Seattle*’ and ‘*San Jose*’ as shown in Figure 17.




john

-- [weather updates](#). - 1 m 46 s ago test 


@wbot what is weather in seattle

[\[Edit\]](#) [\[X\]](#)

---




wbot


-- [weather updates](#). - 1 m 46 s ago test 

The weather is 53 and cloudy in seattle.

---




john

-- [weather updates](#). - 1 m 17 s ago test 


@wbot what is weather in san Jose

[\[Edit\]](#) [\[X\]](#)

---




wbot

-- [weather updates](#). - 1 m 17 s ago test 

The weather is 71 and mostly cloudy in san jose.

*Fig. 18. Conversation between a user (John) and a weather bot (wbot) in experiment 1*



- Admin [Bot Story]

**Account Access**

Manage Account

**Social**

Manage Groups

Feeds and Wikis

Mix Crawls

**Chat Bot**

Bot Story

Intent Entity Expression

### Edit Entity [Add Entity](#)

Entity: location

Entity Value:

San Jose [Delete](#)

chicago [Delete](#)

india [Delete](#)

newyork [Delete](#)

seattle [Delete](#)

Save

Entity Name	Actions
location	<a href="#">Edit</a> <a href="#">Delete</a>
number1	<a href="#">Edit</a> <a href="#">Delete</a>


*Fig. 19. Entity knowledge base of weather bot after experiment 1*

In the Experiment 2, a user tried querying the Weather Bot with a different variation of a sentence which is not in the bot's knowledge base. The conversation between the bot and the user in Experiment 2 is shown in Figure 18.



***Fig. 20. Conversation between a user (John) and a weather bot (wbot) in experiment 2***

The bot recognized intent and entities for the new expression and performed getWeather action and responded with weather information. The Expression knowledge base is updated with the new expression, see Figure 19. And new entity values are stored in Entity knowledge base, see Figure 20.


**- Admin [Bot Story]**

**Account Access**  
Manage Account

**Social**  
Manage Groups  
Feeds and Wikis  
Mix Crawls

**Chat Bot**  
Bot Story

Intent
Entity
Expression

Intent
getWeather


Expression

Bot User has following Entities:  
location  
number1

Save

Expression	Name	Actions
Tell me weather in &location	getWeather	<a href="#">Edit</a> <a href="#">Delete</a>
how is weather in &location	getWeather	<a href="#">Edit</a> <a href="#">Delete</a>
what is the weather in &location	getWeather	<a href="#">Edit</a> <a href="#">Delete</a>

*Fig. 21. Expression Knowledge base of weather bot after experiment 2*


**- Admin [Bot Story]**

**Account Access**  
Manage Account

**Social**  
Manage Groups  
Feeds and Wikis  
Mix Crawls

**Chat Bot**  
Bot Story

Intent
Entity
Expression

Edit Entity
[Add Entity](#)

Entity: location  

San Jose [Delete](#)  
chicago [Delete](#)  
india [Delete](#)  
new jersey [Delete](#)  
newyork [Delete](#)  
seattle [Delete](#)

Entity Value:

Save

Entity Name	Actions
location	<a href="#">Edit</a> <a href="#">Delete</a>
number1	<a href="#">Edit</a> <a href="#">Delete</a>

*Fig. 22. Entity knowledge base after experiment 2*



## CHAPTER 6

### CONCLUSION AND FUTURE WORK

This project is an attempt to design and implement a chatbot framework for Yioop. We have seen how the bot framework is useful for developers to add new chatbots, train them quickly in the Yioop social site. The ultimate aim is to help Yioop developers to create chatbots which are more human like when it interacts with users and make conversational experience more enjoyable.

The end goal of this project is kind of similar to the Facebook bot framework and Microsoft bot framework. The chatbot framework is implemented and integrated with Yioop. This framework provides services to Yioop developers to build, configure and connect bots that can interact with users in a Yioop discussion channel. The development of bots using this framework is so easy, that every developer without understanding of machine learning or artificial intelligence can build more sophisticated bots. Whenever a user sends a message to our bot, he has an intent. For instance, if user types “I want to order a pizza” his intent is “OrderPizza”, “I want to rent a car” intent is “RentCar” etc. Given a sentence, Language understanding API will classify the sentence into one of the trained intents and predicts the top scored intent. The way we achieve this is by defining the intents and training some sentences (called expressions) by manually classifying them using a web based console. The bots can be trained with new scenarios whenever it is required. This API uses Term Frequency and Inverse Document Frequency (TF-IDF) classification algorithm to classify intent. The entities which are necessary information to perform an action, are extracted from the given sentence by using pattern matching technique.

We have tested this framework by creating two example bots: weatherbot and stockbot. The results show that tf\*idf can be used effectively to identify words that typically describe intent of user's request. This outcome shows that the words in the sentence reveals useful information on user's intention. To test the effectiveness, we asked several users to try our chatbot framework and give their opinion. All the users were satisfied and gave a positive feedback by seeing bot learning from the conversations.

As newspapers market, bots are the new apps, chatbots will take over the world in the future and how humans interact with the internet. In the future, every website would have a bot assistant and provide a new way to expose information per need basis of user.

We are excited to provide a preliminary availability of the chatbot Framework for Yioop. This framework can be extended and improved by adding tools or facilities like conversation state management, scheduled conversations, language specific bots and a bot search.

## REFERENCES

- [1] Wit.ai, “docs,” 2016. [Online]. Available: <https://wit.ai/docs>
- [2] developers.facebook.com, “docs,” 2016. [Online]. Available: <https://developers.facebook.com/docs/messenger-platform>
- [3] docs.botframework.com, “core-concepts,” 2016. [Online]. Available: <https://docs.botframework.com/en-us/core-concepts/overview/#navtitle>
- [4] docs.microsoft.com, “LUIS,” 2017. [Online]. Available: <https://docs.microsoft.com/en-us/azure/cognitive-services/LUIS/Home>
- [5] api.slack.com, “bot-users,” [Online]. Available: <https://api.slack.com/bot-users>
- [6] Seekquarry.com, “Resources,” 2015. [Online]. Available: <https://www.seekquarry.com/p/Resources>. [Accessed: 09- Sep- 2015].
- [7] C. Manning, P. Raghavan and H. Schütze, *Introduction to information retrieval*, 1st ed. Cambridge: Cambridge University Press, 2009. [Online]. Available: <https://nlp.stanford.edu/IR-book/>
- [8] Pattern matching. (2017). Retrieved from [https://en.wikipedia.org/wiki/Pattern\\_matching](https://en.wikipedia.org/wiki/Pattern_matching)