ORDER MANAGEMENT CHATBOT

A Project Report

Submitted by

SAI SRIHITHA REDDY THUMMALA
[CB.EN.U4CSE17065]

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

AMRITA SCHOOL OF ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

Amrita Nagar PO, Coimbatore - 641 112, Tamilnadu

May 2021

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, COIMBATORE – 641 112

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **ORDER MANAGEMENT CHAT-BOT** is submitted by **Sai Srihitha Reddy Thummala[CB.EN.U4CSE17065]** in partial fulllment of the requirements for the award of the Degree Bachelor of Technology in Computer Science and Engineering is a bonafide record of the work carried out under our guidance and supervision at Department of Computer Science and Engineering, Amrita School of Engineering, Coimbatore.

PROJECT GUIDE(S)

SIGNATURE

Vidhya

Ms.Vidhya S.
Asst. Prof (Sr. Gr.)
Dept. of Computer Science & Engineering

SIGNATURE

[illegible]

Mr. Supriya Devidutta
Software Engineer Manager
IBM CIO

SIGNATURE

Dr. (Col.) P. N. Kumar
CHAIRPERSON
Dept. of Computer Science & Engineering

June 4, 2021

IBM India Private Limited
Manyata Embassy Business Park,
G2 Block, Nagwara Outer Ring Road,
Bangalore - 560045, India.
Tel : 91-80-49139999
http://www-07.ibm.com/in/careers/

Dear Sai Srihitha Reddy Thummala,

This is to certify that you are presently undergoing an internship training program with IBM India Private Limited from January 13, 2021 to July 12, 2021 under the guidance of Radhesh Kumar Asokan.

Project Title: Order Management Chatbot

Project Details: As B2B deals with many customers , there are many orders, invoices and quotes to be maintained. Often customers and clients require one particular order or list of orders which have same field to check the status or know their details. For this many filters are provided but it is difficult to remember all of them and discern the right filters to get the required information. To make it easier, our task is to create a chatbot using Watson Assistant that handles various queries related to orders (or invoices and quotes) and responds back to the user with the appropriate information that is queried from the database. I have integrated this chatbot with Django backend where user utterances are sent to Watson Assistant using watson python SDK. The web application is dockerized and deployed on to Cirrus Cloud platform.

Internship Serial Number: AVSN39744

Signed By - IBM Authorized Signatory
Director - Talent Acquisition ISA

[illegible]

DECLARATION

I the undersigned solemnly declare that the project report **ORDER MANAGEMENT CHATBOT** is based on my own work carried out during the course of our study under the supervision of Ms.Vidhya S., Asst. Prof (Sr. Gr.), Computer Sciene & Engineering, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgement have been made wherever the findings of others have been cited.

Srihitha.

Sai Srihitha Reddy Thummala[CB.EN.U4CSE17065]

ABSTRACT

International Business Machines (IBM)'s Business-to-Business (B2B) domain deals with many customers, which means they have a huge inventory and an enormous number of orders to maintain for each of them. Each of the orders generate invoices and quotes as they get processed. Orders have fields like Country, Sales Office, Brand, Plant Order Number, Invoice Number etc. They also maintain Quotes, Invoices, Plant Orders etc - which again have various fields. Customers/Clients and IBM Employees frequently look at the orders (or quotes or invoices) for specifications or status. Often, there is a need to filter the massive amount of orders to find a particular order or its related invoice or quote. For this, IBM provides many filters based on the fields. The user needs to have knowledge about what filters are available and where to find them. This might take many clicks if you are not familiar with them.

To make this process easy, a chatbot is created and deployed on cloud - which can take the client/user's queries in English and return the appropriate information back, again in English. This will abstract the whole process of getting the right filters, querying the database and formatting the output.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to our beloved Satguru **Sri Mata Amritanandamayi Devi** for providing the bright academic climate at this university, which has made this entire task appreciable. This acknowledgement is intended to be a thanksgiving measure to all those people involved directly or indirectly with my project. I would like to thank our Pro Chancellor **Bramachari Abhayamrita Chaitanya**, Vice Chancellor **Dr. Venkat Rangan. P** and **Dr. Sasangan Ramanathan**, Dean Engineering of Amrita Vishwa Vidyapeetham for providing us the necessary infrastructure required for the completion of the project. I express my thanks to **Dr. (Col) P.N. Kumar**, Chairperson of Department of Computer Science Engineering, **Dr. G. Jeyakumar and Dr. C. Shunmuga Velayutham,** Vice Chairpersons of the Department of Computer Science and Engineering for their valuable help and support during our study. I express my gratitude to my guide, **Ms. Vidhya S.**, Assistant Professor (Sr. Gr.), Department of Computer Science and Engineering for her guidance, support and supervision. I feel extremely grateful to **Ms. Anisha Radhakrishnan**, Assistant Professor, Department of Computer Science and Engineering, **Dr. R. Karthi**, Associate Professor, Department of Computer Science and Engineering and **Dr. K. Raghesh Krishnan**, Assistant Professor (Sr. Gr.), Department of Computer Science and Engineering for their feedback and encouragement which helped us to complete the project. I would also like to thank the entire staff of the Department of Computer Science and Engineering. I would like to extend my sincere thanks to my family and friends for helping and motivating me during the course of the project. Finally, I would like to thank all those who have helped, guided and encouraged me directly or indirectly during the project work. Last but not the least, I thank God for His blessings which made my project a success.

Srihitha.

Sai Srihitha Reddy Thummala

CB.EN.U4CSE17065

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

B2B	Business-to-Business
IBM	International Business Machines
CIO	Chief Information Office
Q2C	Quote to Cash
AI	Artifical Intelligence
API	Application Programming Interface
WSL	Windows Subsystem for Linux
UI	User Interface
SDK	Software Development Kit
OData	Open Data Protocol
IAM	IBM cloud Identity and Access Management
JSON	JavaScript Object Notation

Chapter 1

INTRODUCTION

Chatbot is an Artifical Intelligence (AI) software agent that stimulates conversation in Natural Language with the users automatically based on predefined triggers. They offer new oppportunities to engage customers. They can help users in various tasks from booking tickets to answering some of the common questions. They empower business and save time by answering User queries 24x7. Chatbots are convenient as they provide operational and economical efficiency to the companies. Because of this, they are being used in a wide variety of domains, including Business, Customer Service, Real Estate, Health, News etc. (Botscrew, Botscrew)

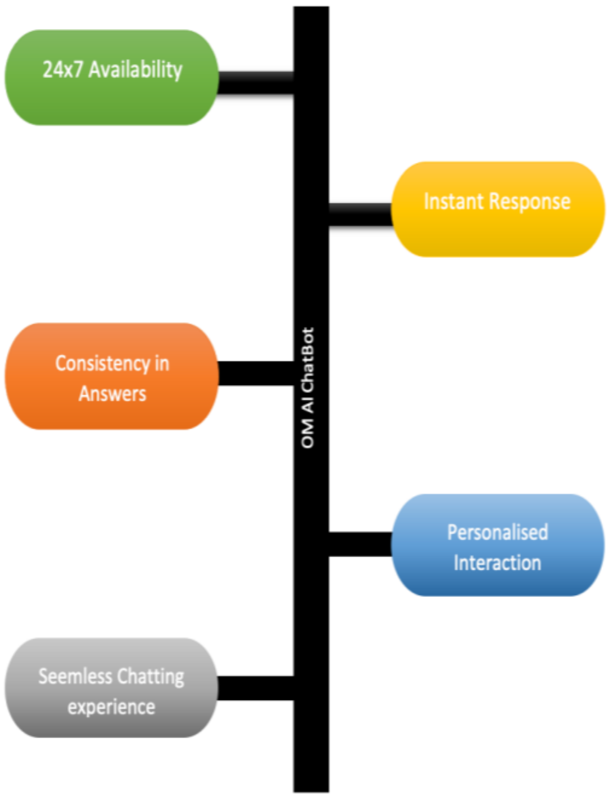


Figure 1.1: Benefits of Chatbot

B2B Gateway Services of IBM Chief Information Office (CIO) deals with customers, suppliers, business partness and internal stackeholders to efficiently and effectively interact with IBM's worldwide Quote to Cash (Q2C) and Supply Chain applications. As B2B deals with many customers, they have a huge inventory and an enormous quantity of orders to maintain. Each of the orders generate invoices and quotes as they get processed. Users frequently look at the orders (or quotes or invoices) for specifications or status.

1.1 Problem Definition

To find a particular order, the user needs to filter through the large amount of invoices, quotes and orders stored. For this, the user should have the knowledge of various filters available and know which filter would give him the respective data. Chatbots can mitigate the frustration of searching for the right filter and improve the user experience. Users can converse with the chatbot to get the relevant data almost immediately instead of figuring out which filters to use.
The high level pipeline of the project is as follows:

1. To train Watson Assistant with appropriate intents and entities
2. To integrate the chatbot into a web application
3. To deploy it onto Cirrus (IBM Cloud service)

Chapter 2

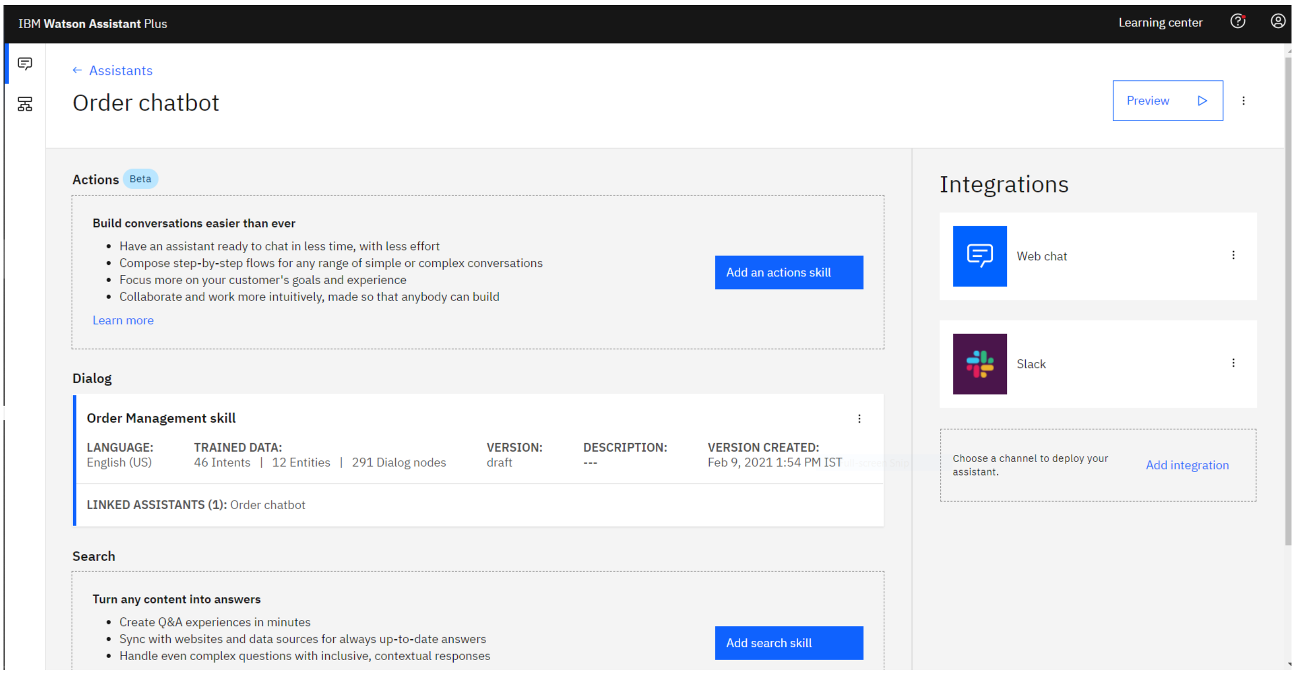
EXISTING SYSTEMS

2.1 Watson Assistant

AI is becoming increasingly smarter and readily available to everyone these days. Such cognitive capabilities are now easily available through services provided by companies. Watson Assistant is one such AI service on the IBM Cloud provided by IBM to help build and deploy virtual assistants. Watson is readily available and is equipped with Application Programming Interface (API)s that are available to anyone with a cloud account.

2.1.1 Dialogue Skill

Assistants are equiped with **Dialogue Skill** (Figure 2.1) to converse with the user. A dialog skill uses Watson Natural Language Processing and Machine Learning technologies to understand user questions and requests, and respond to them with answers. To create a fully functional chatbot, a skill needs intents and entities to train the assistant.



Order chatbot

Actions

Build conversations easier than ever

- Have an assistant ready to chat in less time, with less effort
- Compose step-by-step flows for any range of simple or complex conversations
- Focus more on your customer's goals and experience
- Collaborate and work more intuitively, made so that anybody can build

Learn more

Dialog

Order Management skill

LANGUAGE:	TRAINED DATA:	VERSION:	DESCRIPTION:	VERSION CREATED:
English (US)	46 Intents | 12 Entities | 291 Dialog nodes	draft	---	Feb 9, 2021 1:54 PM IST

LINKED ASSISTANTS (1): Order chatbot

Search

Turn any content into answers

- Create Q&A experiences in minutes
- Sync with websites and data sources for always up-to-date answers
- Handle even complex questions with inclusive, contextual responses

Choose a channel to deploy your assistant.

Figure 2.1: Assistant linked to a dialogue skill

In a high level, Figure 2.2 depicts how the chatbot works. The Assistant uses 3 main components - intents, entites and dialogue to determine how to interpret the user input and how to respond to it.

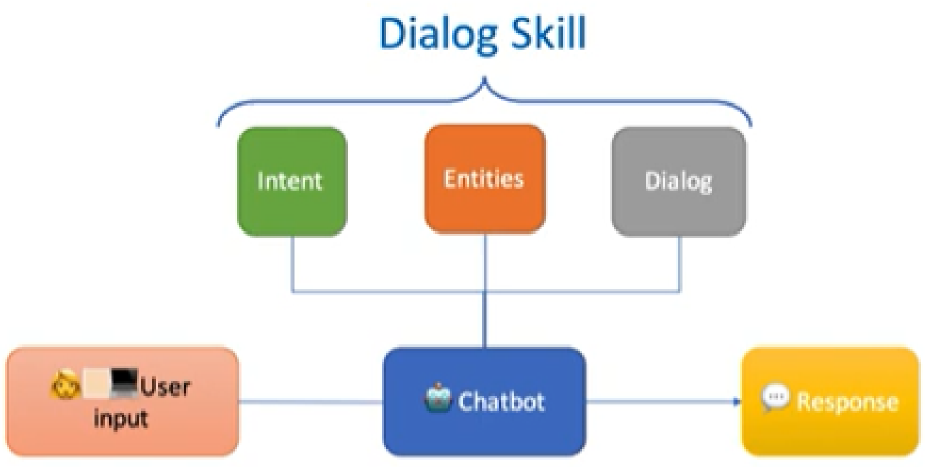
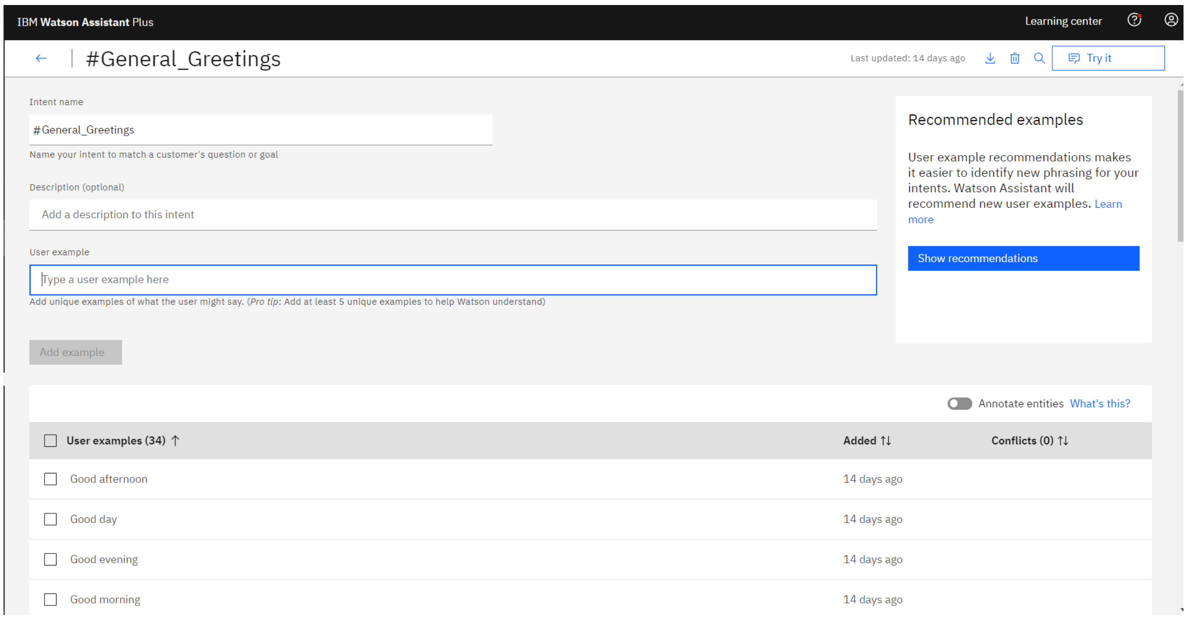


Figure 2.2: High level depiction of Watson Assistant

2.1.2 Intents and Entities

When a user queries a chatbot, the chatbot tries to classify the user's intention based on the pretrained **intents**. In other words, Intents try to determine what the user wants, what are they asking for and capture the intent or the goal of the user.
Each intent is trained with a set of examples which stay true to a common intention. As seen in figure 2.3, the intent General_Greetings is trained with various examples such as "Hi", "Good morning", "Hello" etc., to indicate the intention of greeting the bot.


IBM Watson Assistant Plus | Learning center

#General_Greetings

Last updated: 14 days ago | Try it

Intent name

#General_Greetings

Name your intent to match a customer's question or goal

Description (optional)

Add a description to this intent

User example

Type a user example here

Add unique examples of what the user might say. (*Pro tip*: Add at least 5 unique examples to help Watson understand)

Add example

Recommended examples

User example recommendations makes it easier to identify new phrasing for your intents. Watson Assistant will recommend new user examples. Learn more

Show recommendations

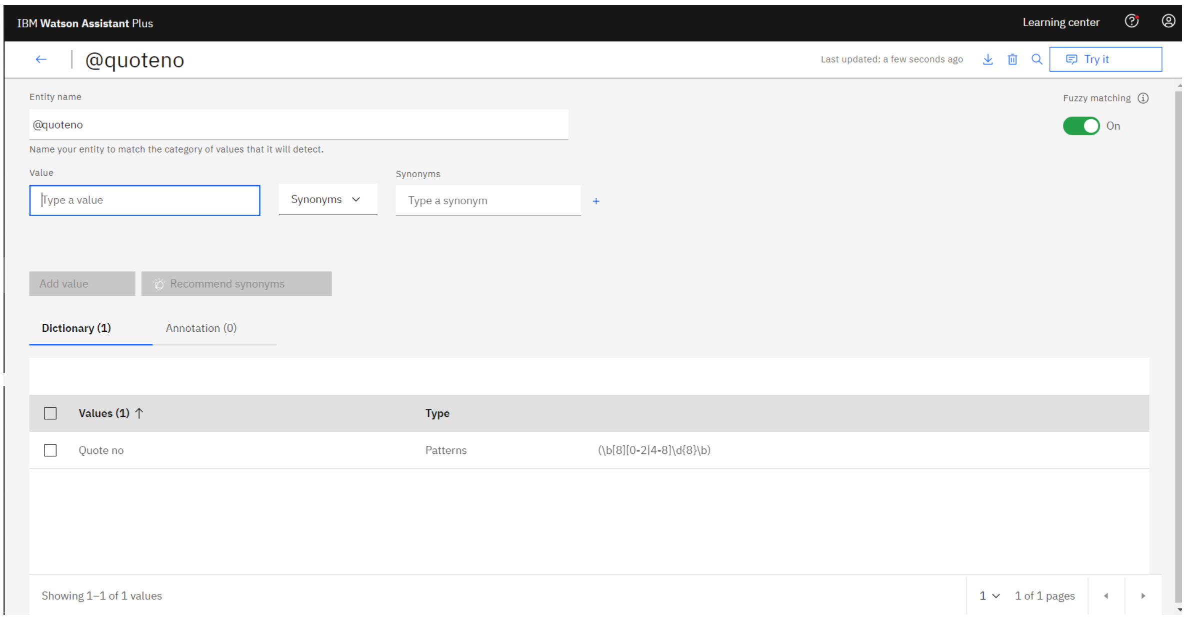
Annotate entities What's this?

User examples (34) ↑	Added ↑↓	Conflicts (0) ↑↓
Good afternoon	14 days ago	
Good day	14 days ago	
Good evening	14 days ago	
Good morning	14 days ago	

Figure 2.3: Example Intent General_Greetings

Entities capture potential input information. They are specific information provided by the users which can be used to answer the input query.

Entities can be trained by adding multiple words and their synonyms. They can also be defined by a pattern with the help of Regular Expressions. As shown in Figure 2.4, the entity *quote* is trained with a regular expression which captures a 10 digit number starting with 8 but not 83 or 89.


IBM Watson Assistant Plus | Learning center

@quoteno

Last updated: a few seconds ago | Try it

Entity name

@quoteno

Name your entity to match the category of values that it will detect.

Fuzzy matching: On

Value: Type a value | Synonyms | Synonyms: Type a synonym

Add value | Recommend synonyms

Dictionary (1) | Annotation (0)

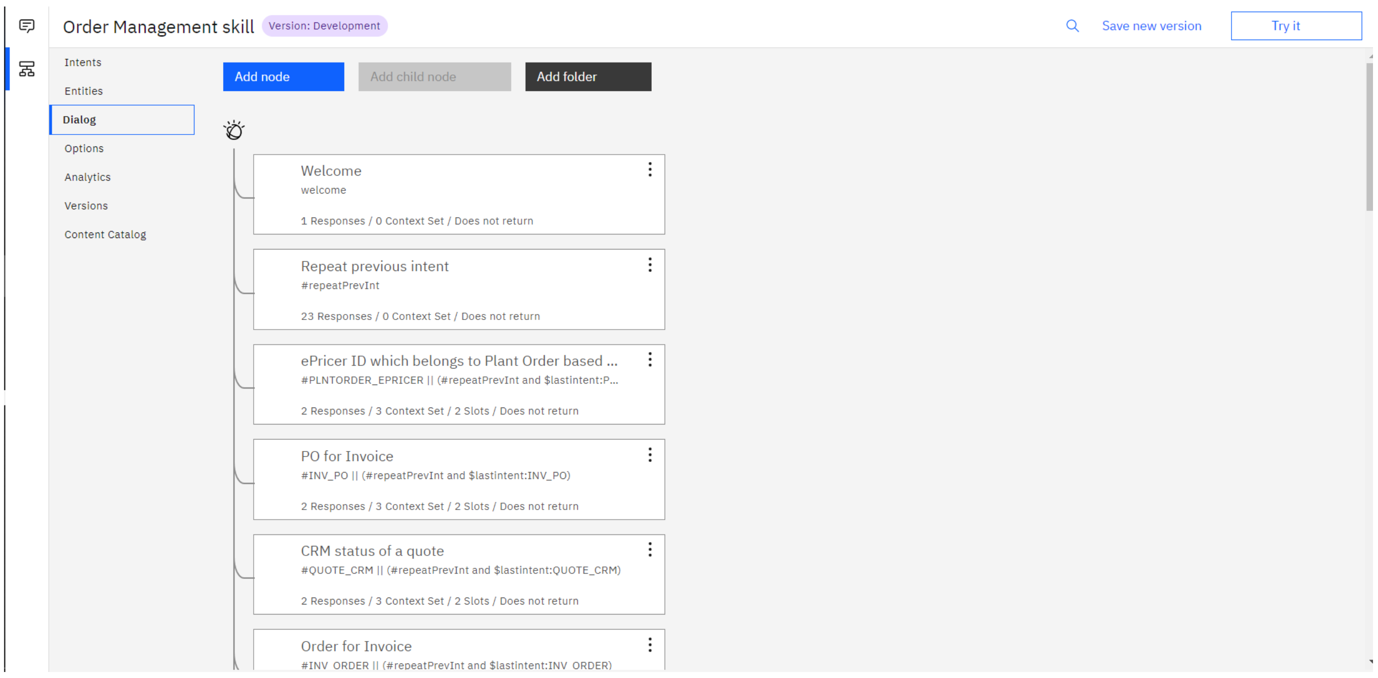
Values (1) ↑	Type	
Quote no	Patterns	(\b[8][0-2|4-8]\d{8}\b)

Showing 1–1 of 1 values | 1 | 1 of 1 pages

Figure 2.4: Quote Entity

2.1.3 Dialogue

The Dialogue Flow (Figure 2.5) helps formulate responses to the user queries and direct the conversation.


Order Management skill Version: Development

Save new version Try it

Intents
Entities
Dialog
Options
Analytics
Versions
Content Catalog

Add node Add child node Add folder

Welcome
welcome
1 Responses / 0 Context Set / Does not return

Repeat previous intent
#repeatPrevInt
23 Responses / 0 Context Set / Does not return

ePricer ID which belongs to Plant Order based ...
#PLNTORDER_EPRICER || (#repeatPrevInt and $lastintent:P...
2 Responses / 3 Context Set / 2 Slots / Does not return

PO for Invoice
#INV_PO || (#repeatPrevInt and $lastintent:INV_PO)
2 Responses / 3 Context Set / 2 Slots / Does not return

CRM status of a quote
#QUOTE_CRM || (#repeatPrevInt and $lastintent:QUOTE_CRM)
2 Responses / 3 Context Set / 2 Slots / Does not return

Order for Invoice
#INV_ORDER || (#repeatPrevInt and $lastintent:INV_ORDER)

Figure 2.5: Dialogue flow of the Skill

Dialogue flow consists of **Nodes** (Figure 2.6) which are processed when the related conidtion is triggered. Conditions (Figure 2.7) can be entities and intents recognized or even a combination of them. A **Dialogue** is a simple tree of nodes where each node typically handles a scenario.

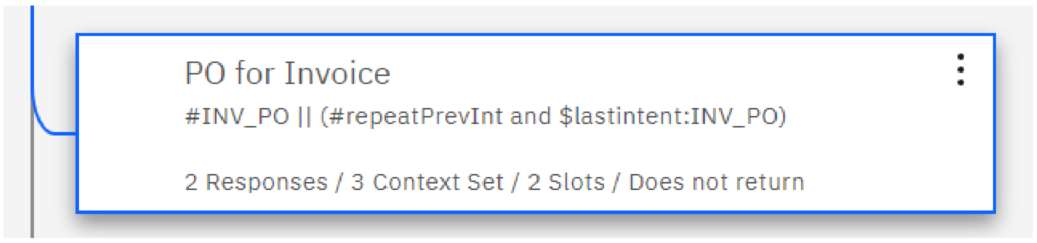
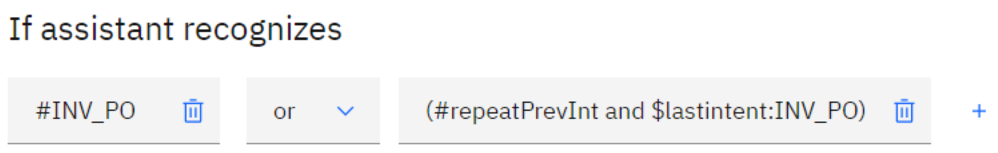

PO for Invoice
#INV_PO || (#repeatPrevInt and $lastintent:INV_PO)
2 Responses / 3 Context Set / 2 Slots / Does not return

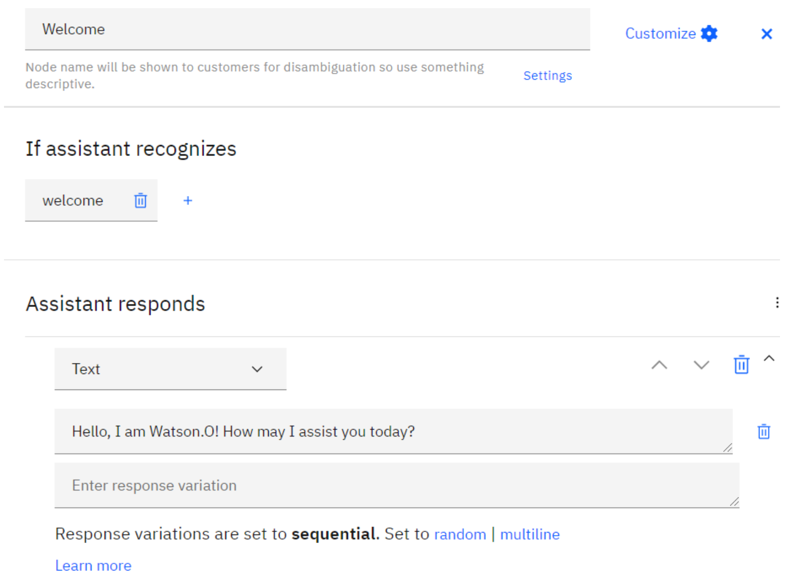
Figure 2.6: A Node


If assistant recognizes

#INV_PO or (#repeatPrevInt and $lastintent:INV_PO)

Figure 2.7: Conditions to be met to process the Node in Fig.2.6

Two nodes - *Welcome* (Figure 2.8 (a)) and *Anything else* (Figure 2.8 (b)) are automatically created for a skill in the dialogue. Welcome Node is the first node and it greets the user when they engage with the assistant. The Anything else node is the final node in the dialogue tree and is triggered when the user input is not recognized. Any other node sits between these two nodes.



Welcome

Node name will be shown to customers for disambiguation so use something descriptive.

If assistant recognizes

welcome

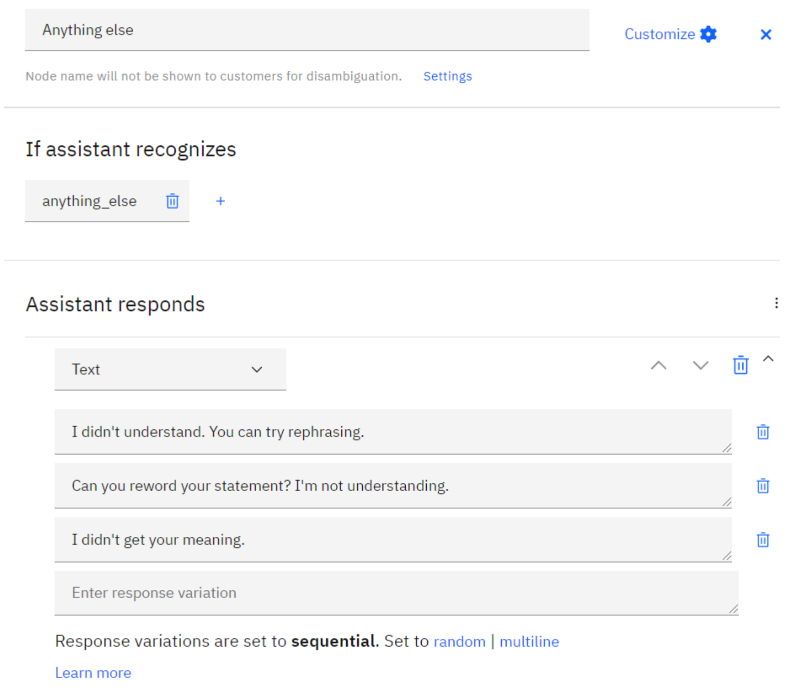
Assistant responds

Hello, I am Watson.O! How may I assist you today?

Enter response variation

Response variations are set to **sequential**. Set to random | multiline

(a) Welcome node



Anything else

Node name will not be shown to customers for disambiguation.

If assistant recognizes

anything_else

Assistant responds

I didn't understand. You can try rephrasing.

Can you reword your statement? I'm not understanding.

I didn't get your meaning.

Enter response variation

Response variations are set to **sequential**. Set to random | multiline

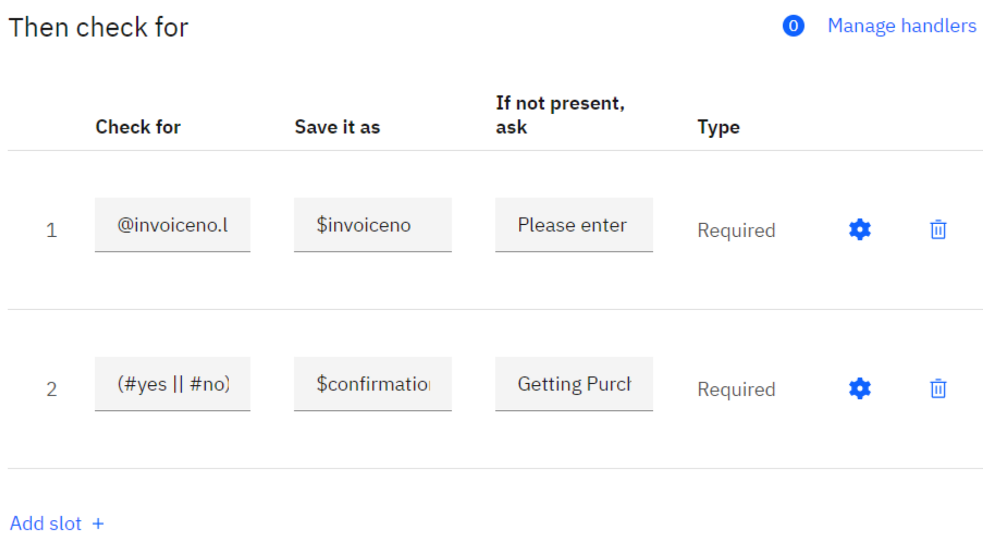
(b) Aything else node

Figure 2.8: The first and last nodes of a dialogue tree

The execution of the user input is evaluated from the top to the bottom of the dialogue. When the user submits their utterance, the first node will be evaluated for execution. If the condition is not met, the second node is then evaluated and so on. Essentially, the nodes are continued to be evaluated in the order they appear in the dialogue tree while the conditions of these nodes are false. Watson stops at the first node whose conditions matches the user input.

This is also the reason, the fall back node is at the very bottom. This node has a special condition that is always true. So whenever all the above nodes fail to execute, this node is automatically triggered. It ensures a reply for the user. For a similar reason, the welcome node is present at the very top. This node has a special condition which is true at the beginning of the conversation with the user. This condition is only true once, which ensures that the user is not greeted every time an input is evaluated.

Sometimes, the entities are necessary to answer the user query. If the user does not provide the required information, the user can be prompted to give the entity explicitly. Watson Assistant provides a feature called **slots** (Figure 2.9) for this. Each node can be tuned to have slots to gather that additional information.



Then check for

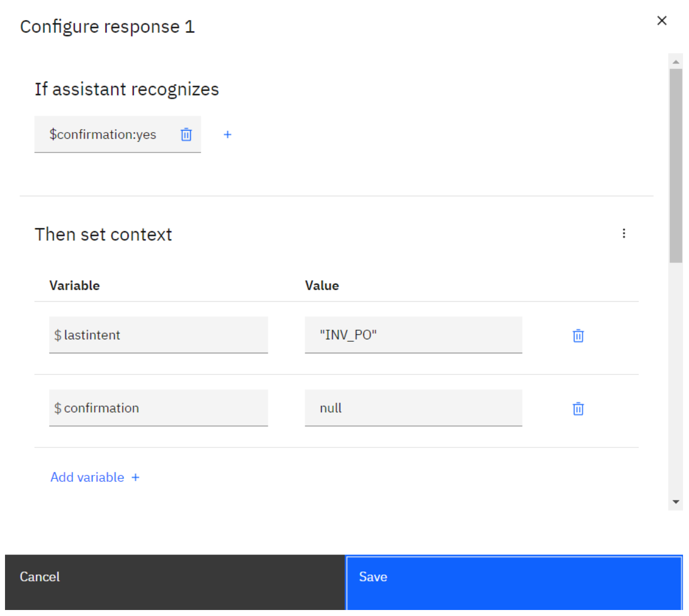
0 Manage handlers

	Check for	Save it as	If not present, ask	Type
1	@invoiceno.l	$invoiceno	Please enter	Required
2	(#yes || #no)	$confirmatio	Getting Purcl	Required

Add slot +

Figure 2.9: Slot configuration in a Node

Life cycle of the entities is usually only one dialogue. Having a context makes the chatbot smarter as it can remember it in the next dialogue. A **Context Variable** (Figure 2.10) is a variable that is defined in a node. Other nodes, application logic, or user input can subsequently set or change the value of the context variable. Unlike entities, context variables remain available for the entire duration of the conversation with the user, so the chatbot can refer back to them anytime. They can be used to retain the information provided by the user across dialogue turns, through the conversation. Often, the information gathered through the slots is stored in context variables. This can make the bot seem more genuine and help make it easy for the user to chat with it.


Configure response 1

If assistant recognizes

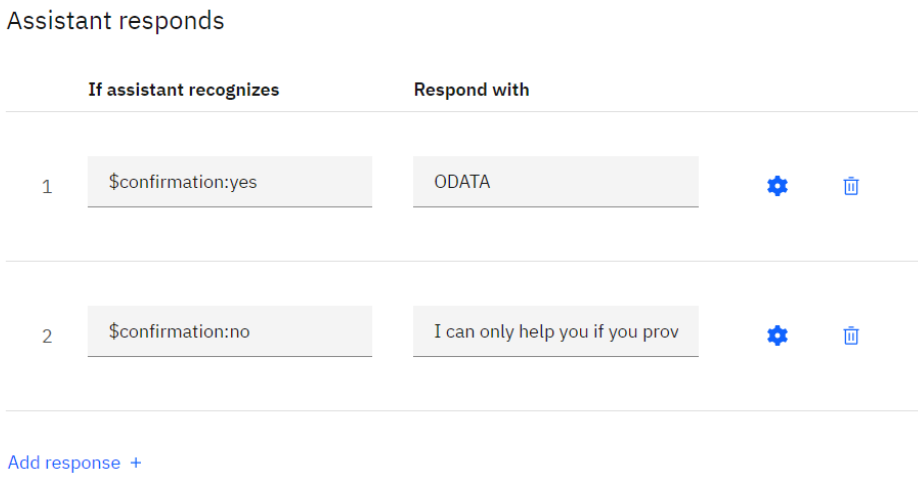
$confirmation:yes

Then set context

Variable	Value
$ lastintent	"INV_PO"
$ confirmation	null

Figure 2.10: Setting context variables in the response of a Node

When a node is finally processed, Watson can be set to respond in multiple ways based on the condition set in place. The condition can be an intent, entity or even a context variable. Figure 2.11 shows responses being configured based on a context variable named confirmation.


Assistant responds

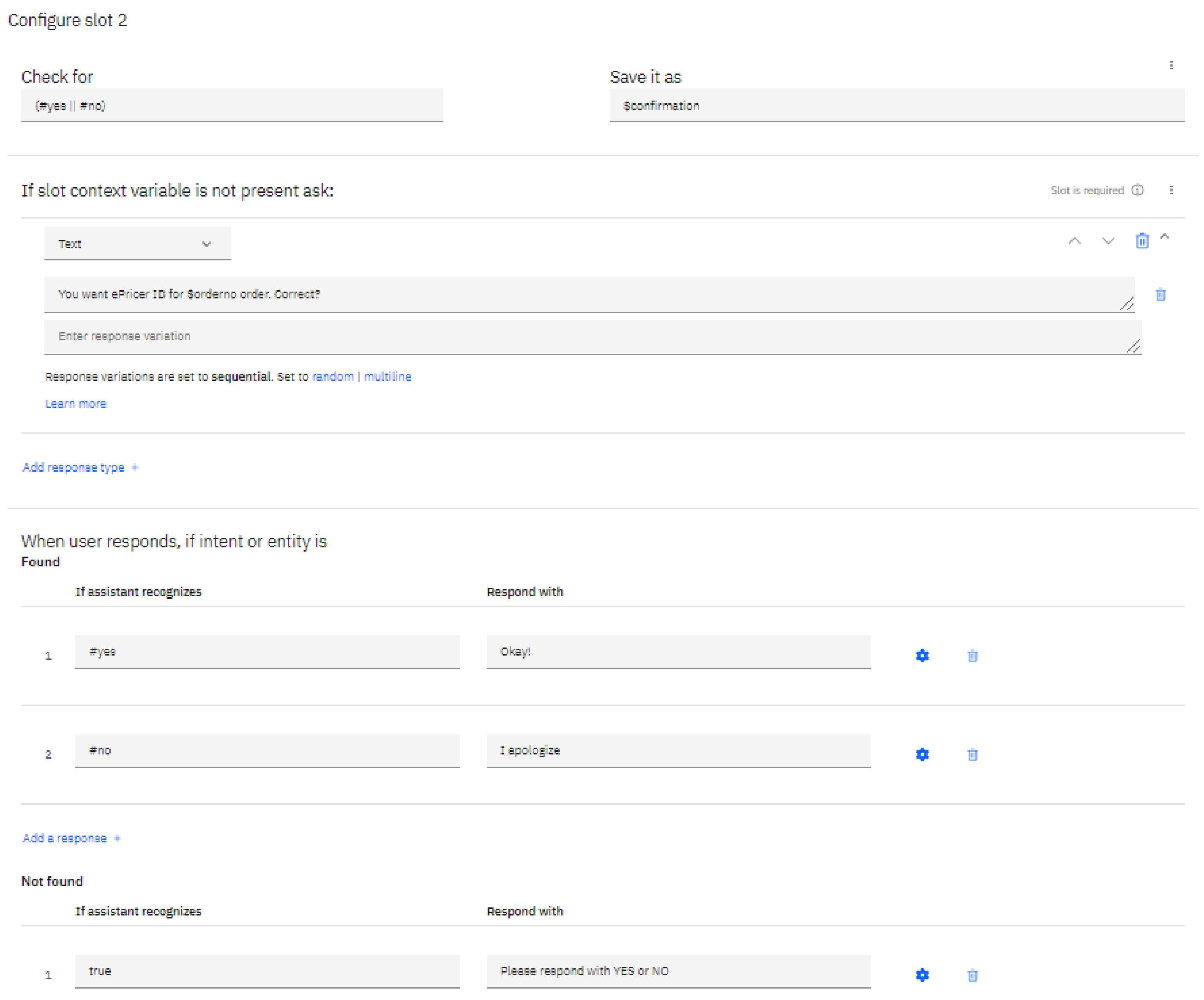
	If assistant recognizes	Respond with
1	$confirmation:yes	ODATA
2	$confirmation:no	I can only help you if you prov

Figure 2.11: Multiple responses configured for a Node

2.1.4 Digressions

Required slots don't quit asking their question until an acceptable answer has been provided and stored in a context variable. This can be an issue as the bot can seem awkward and obsessive if the user wants to break away from the conversation.

This can be avoided in two ways. One way is to configure the *Found* and *Not found* sections (Figure 2.12) to mitigate this issue. These sections allow to specify what the slot will say to the end user when they reply with an acceptable answer and when they don't respectively. Found section is typically used to thank the user. The not found section can be used to make asking the same question again less awkward.



Configure slot 2

Check for	Save it as
(#yes || #no)	$confirmation

If slot context variable is not present ask:

Slot is required

Text

You want ePricer ID for $orderno order. Correct?

Enter response variation

Response variations are set to **sequential**. Set to random | multiline

Learn more

Add response type +

When user responds, if intent or entity is

Found

	If assistant recognizes	Respond with
1	#yes	Okay!
2	#no	I apologize

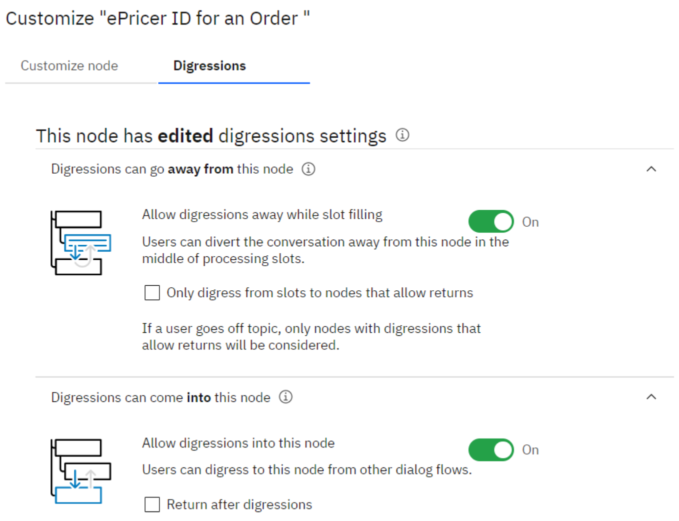
Add a response +

Not found

	If assistant recognizes	Respond with
1	true	Please respond with YES or NO

Figure 2.12: Found and Not Found Sections in a Slot

Another way is to enable **Digressions**. Digressions (Figure 2.13) allow a slot to jump to a different question after having asked the mandatory question. Settings can be enabled in such a way that other nodes can return back to the orginating slot after answering the user's side question. It can also be set to not go back to the originating slot and conitnue the conversation from the node Watson jumped to.



Customize "ePricer ID for an Order "

This node has **edited** digressions settings

Digressions can go **away from** this node

Allow digressions away while slot filling — On

Users can divert the conversation away from this node in the middle of processing slots.

☐ Only digress from slots to nodes that allow returns

If a user goes off topic, only nodes with digressions that allow returns will be considered.

Digressions can come **into** this node

Allow digressions into this node — On

Users can digress to this node from other dialog flows.

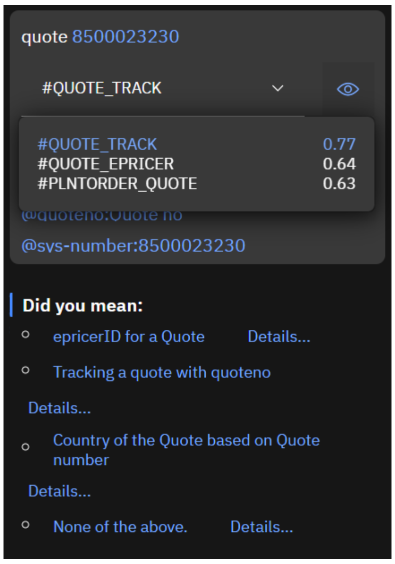
☐ Return after digressions

Figure 2.13: Settings to enable Digressions

2.1.5 Autolearning and Disambiguation

When the Assistant associates a user query with an intent, it assigns some confidence score to it. The assistant shows the top 3 intents with their confidence scores when a user submits their query.

When the confidence scores of the runner up intents that are detected in the user input are close in value to the confidence score of the top intent, the **disambiguation** is triggered. That is, it is triggered when there is no clear separation and certainity. There is a secondary condition to be met that the confidence score of the top intent is above 0.2. Figure 2.14 shows the top 3 intents along with their confidence scores for user input "Quote 8500023230". The top intent and the runner up intents have close confidence scores which triggered disambiguation.



quote 8500023230

#QUOTE_TRACK

#QUOTE_TRACK	0.77
#QUOTE_EPRICER	0.64
#PLNTORDER_QUOTE	0.63

@quoteno:Quote no

@sys-number:8500023230

Did you mean:

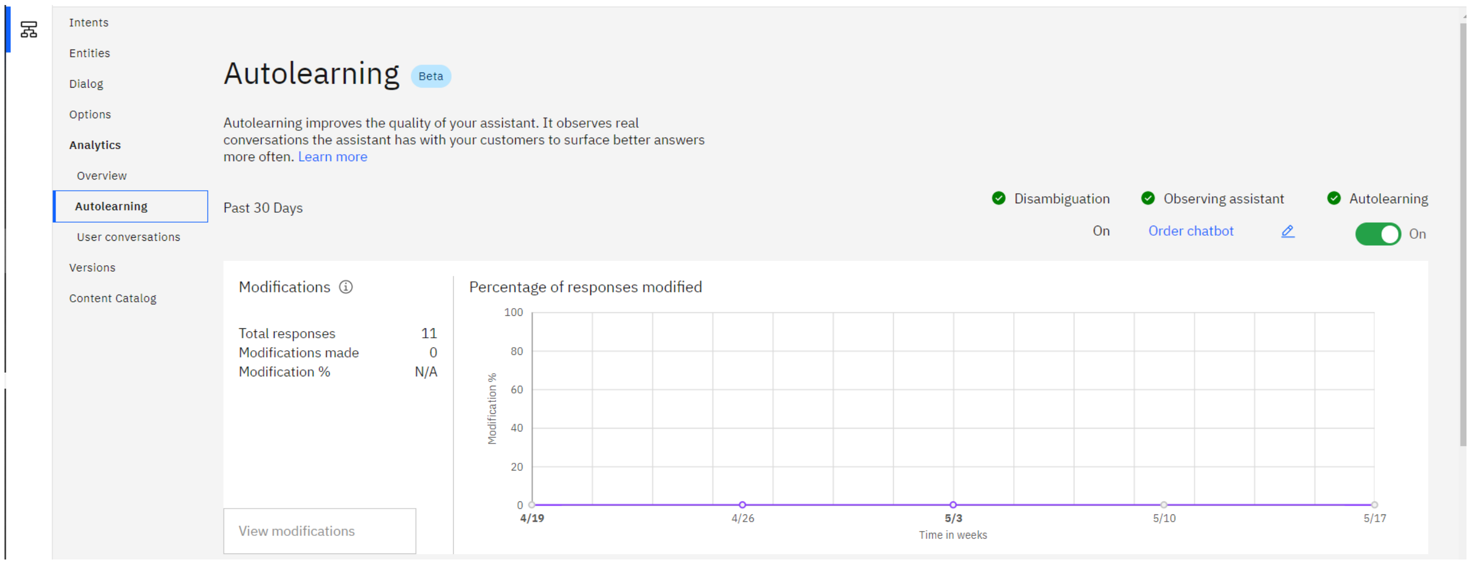
- epricerID for a Quote Details...
- Tracking a quote with quoteno Details...
- Country of the Quote based on Quote number Details...
- None of the above. Details...

Figure 2.14: Disambiguation Example

A list of related node names are presented as options to the user, allowing the user to disambiguate the dialog by selecting an option from the list. Disambiguation allows the chatbot to request clarification from the user.

The nodes in the dialogue tree can be set in such a way that a particular node is given priority over disambiguation. A node can also be set to be hidden away from disambiguation. This will not show the node in the list of options when disambiguation is triggered. It is important that the name of the node displayed to the user is clear, presentable and explains the function and intention of the node it refers to.

Watson Assistant also provides the option of **Autolearning** (Figure 2.15). In short, the chatbot learns from interactions with the users. Autolearning improves the quality of the skill over time. It applies insights that it gains from observing user interactions during disambiguation to help the skill identify and surface correct answers more often. Autolearning only works when disambiguation is activated and in use. It learns from the user interactions with the menu presented to dissolve the dismabiguation. Watson Assistant automatically adjusts the menu sequence based on popularity. It is worth noting that the options of the menu can be different on entering the same user input consecutively.



Autolearning

Autolearning improves the quality of your assistant. It observes real conversations the assistant has with your customers to surface better answers more often.

Past 30 Days

Disambiguation: On
Observing assistant: Order chatbot
Autolearning: On

Modifications

Total responses	11
Modifications made	0
Modification %	N/A

Figure 2.15: Autolearning in Watson Assistant

2.2 Software/Tools Requirements

Software and Libraries used in this project are -

1. Watson Assistant (Plus Version) – IBM Chatbot AI Service
2. ibm-watson – Python library for Watson APIs
3. ibm-cloud-sdk-core - Python library to Authenticate for Watson
4. Django – A high-level Python Web framework
5. Docker with Windows Subsystem for Linux (WSL) – Containerization tool
6. Cirrus – IBM Cloud Service
7. Bootstrap – CSS framework
8. Open Data Protocol (OData) – REST-based protocol for querying and updating data

Chapter 3

PROPOSED SYSTEM

3.1 System Analysis

The **ORDER MANAGEMENT CHATBOT** project can be broken down into the following modules.

S.No	Module
1	Training Watson Assistant
2	Creating a web application to deploy chatbot
3	Connecting the web application to Watson Assistant
4	Connecting the web application to the database
5	Toggling Disambiguation and Autolearning
6	Dockerizing the web application
7	Deploying the container onto Cirrus
8	Integrating the Cirrus application with LogDNA

Table 3.1: Module details

3.1.1 System requirement analysis

To support ibm-watson package and Django framework, Python 3.6 or above is required. And for Docker with WSL, the following requirements are to be met:

- Operating System - Windows 10 (ver 1903 or higher) 64-bit
- 4GB RAM
- BIOS Level virtualization support

In this project, Bootstrap v4.5 is used. It is supported on all latest major browsers (Bootstrap, Bootstrap).

3.1.2 Module details of the system

Training Watson Assistant

The Watson Assistant is trained with appropriate intents and entities. The confidence score of the top intent recognized should be more than 0.2 so the user can receive a response from Watson. The features provided, such as - slots, context variables, multiple responses and digressions are used to maximize the effectiveness of the bot in responding to the user. By depending on Watson Assistant to classify the user utterance, the backend can mainly be focused on connecting the user and watson and also providing the data from the database in the response.

Creating a web application to deploy chatbot

The python web framework - Django is used to create a web application. A simple User Interface (UI) is built with the help of Bootstrap to help visualize the bot. A chat icon and a chatbox are built so the user can interact with Watson by entering the user input into the textbox. The bot responses are also shown in the chatbox.

Connecting the web application to Watson Assistant

In the previous module, the frontend containing the chatbox is built. But the user input is taken from the frontend and to the backend so it can be sent to the Watson Assistant. The python Software Development Kit (SDK) ibm-cloud-sdk-core can be used to make authenticated requests to Watson. Here, another python library for Watson API, ibm-watson is used to send the message to the Watson Assistant. The Assistant API provides runtime methods that the client application can use to send user input to an assistant and receive a response.

Connecting the web application to the database

Once the response is received from the Watson Assitant, the intents and entities recognized can be gathered from it. Based on them, suitable data is to be requested from the database and parsed with the bot response and passed to the user. IBM uses a SAP database to store all the orders, quotes and invoices. To access the data, an OData request URL is provided. With appropriate paramaters, the data is brought back to the backend and parsed with the bot response for the user.

Toggling Disambiguation and Autolearning

To enhance the chatbot further, Disambiguation and Autolearning are enabled. To acco-modate the disambiguation feature, the backend has to be able to handle the suggestion type response from the chatbot.

Dockerizing the web application

To deploy it onto Cirrus, the web application has to be dockerized. A dockerfile along with a docker-compose yaml file is written. The dockerfile contains all the dependencies needed for the chatbot to be assembled as an image. The docker-compose file is used to configure the docker container.

Deploying the container onto Cirrus

An internet facing project is created on Cirrus, IBM's private cloud service. A pipeline is then created to which the docker image is added. This pipeline is used to create an application with a custom URL to access the web application over the internet. Related firewall settings are to be configured to successfully run the chatbot.

Integrating the Cirrus application with LogDNA

For any software project created, it is important to log from a security point of view. It can help point out the red flag when something wrong is happening. Cirrus allows the projects to be integrated with LogDNA, a log management solutoin to save the logs of the deployed applications.

3.2 System Design

3.2.1 Flow diagram of the system

Figure 3.1 shows top level Data flow diagram with components of the application.

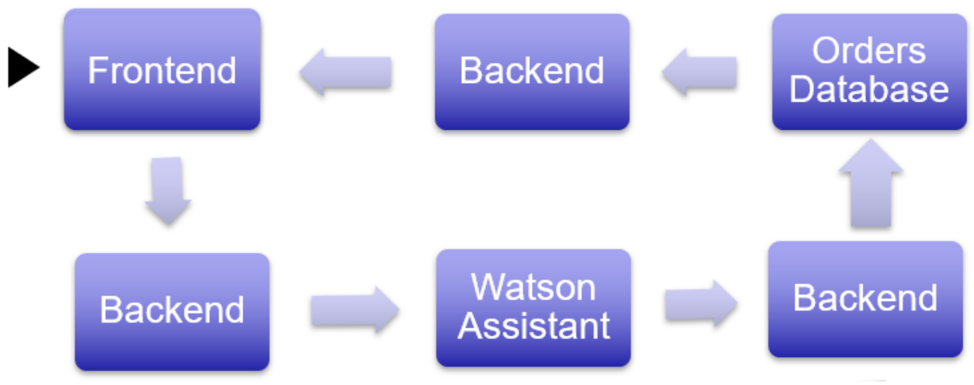


Figure 3.1: Flow Diagram

When the user enters the query in the chatbox, the query is brought ot the backend where Watson SDK sends the message to Watson Assistant. Based on the response received from Watson Assisntant, appropriate data is queried from the orders database through OData API. The data and the resposne are then parsed in the backend and sent to the frontend for the user to see. This is the data flow cycle of one dialogue between the user and the Watson Assistant.

3.2.2 Architecture diagram

Figure 3.2 shows the architecture diagram of the web application.

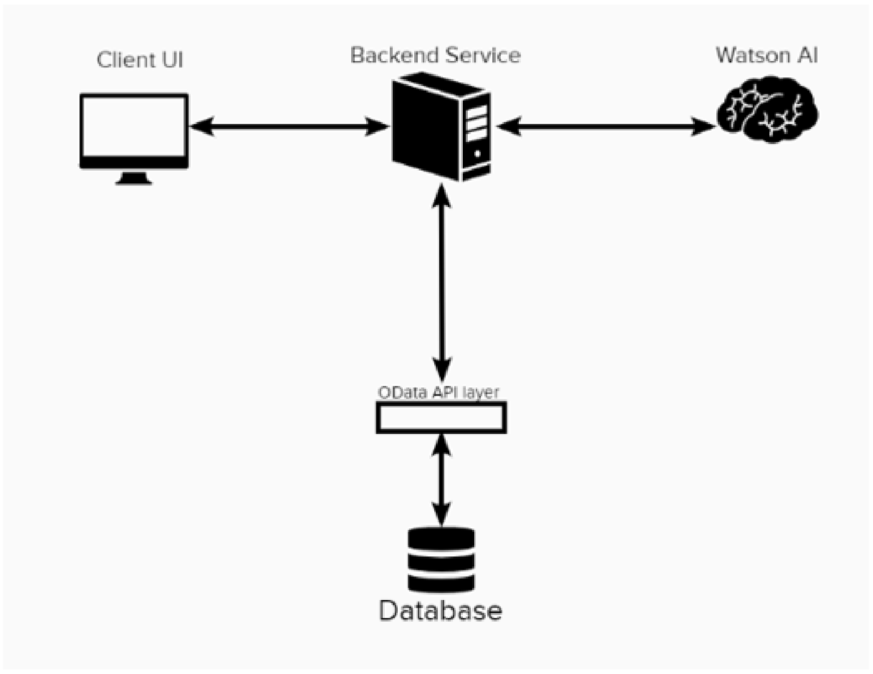


Figure 3.2: Architecture Diagram

The client UI is connected to the backend. The backend service communicates with both the Watson AI and OData API to formulate a response for the user and sends it back to the client UI.

Chapter 4

IMPLEMENTATION AND TESTING

4.1 Training the Watson Assistant

As explained in Section 2.1.1, the Watson Assistant is trained with relevant Intents and Entities so it can respond to the user. Currently, the chatbot is trained to respond to the following 17 scenarios.

- QUOTE_FOP - Here, the input is Quote ID and the output is FOP status of the quote.
  Ex: proivde fop status for Quote xxxxxx
- QUOTE_CRM - Here, the input is Quote ID and the output is CRM status of the quote.
  Ex: crm status for Quote xxxxxx ?
- QUOTE_TRACK - Here, the input is Quote ID and the output is Quote details.
  Ex: Please get me tracking details for Quote xxxxxx
- QUOTE_STATUS - Here, the input is Quote ID and the output is status description of the quote.
  Ex: Show me Status of Quote xxxxxx
- QUOTE_COUNTRY - Here, the input is Quote ID and the output is Country, sales office details of the quote.
  Ex: Get me the Country details of Quote xxxxxx ?
- QUOTE_EPRICER - Here, the input is Quote ID and the output is epricer ID.
  Ex: which epricer ID is assigned to Quote xxxxxxx ?
- ORDER_STATUS - Here, the input is Sales Order ID and output is Status description of the order.
  Ex: what is the status for order xxxxxx ?
- ORDER_TRACK - Here, the input is Sales Order ID and output is order details.
  Ex: I want details of sales order xxxxxx ?
- ORDER_EPRICER - Here, the input is Sales Order ID and output is epricer ID.
  Ex: what epricer ID belongs to Sales Order xxxxxx ?
- ORDER_COUNTRY - Here, the input is Sales Order ID and output is Country, Sales office details.
  Ex: which country contains sales order xxxxxx ?

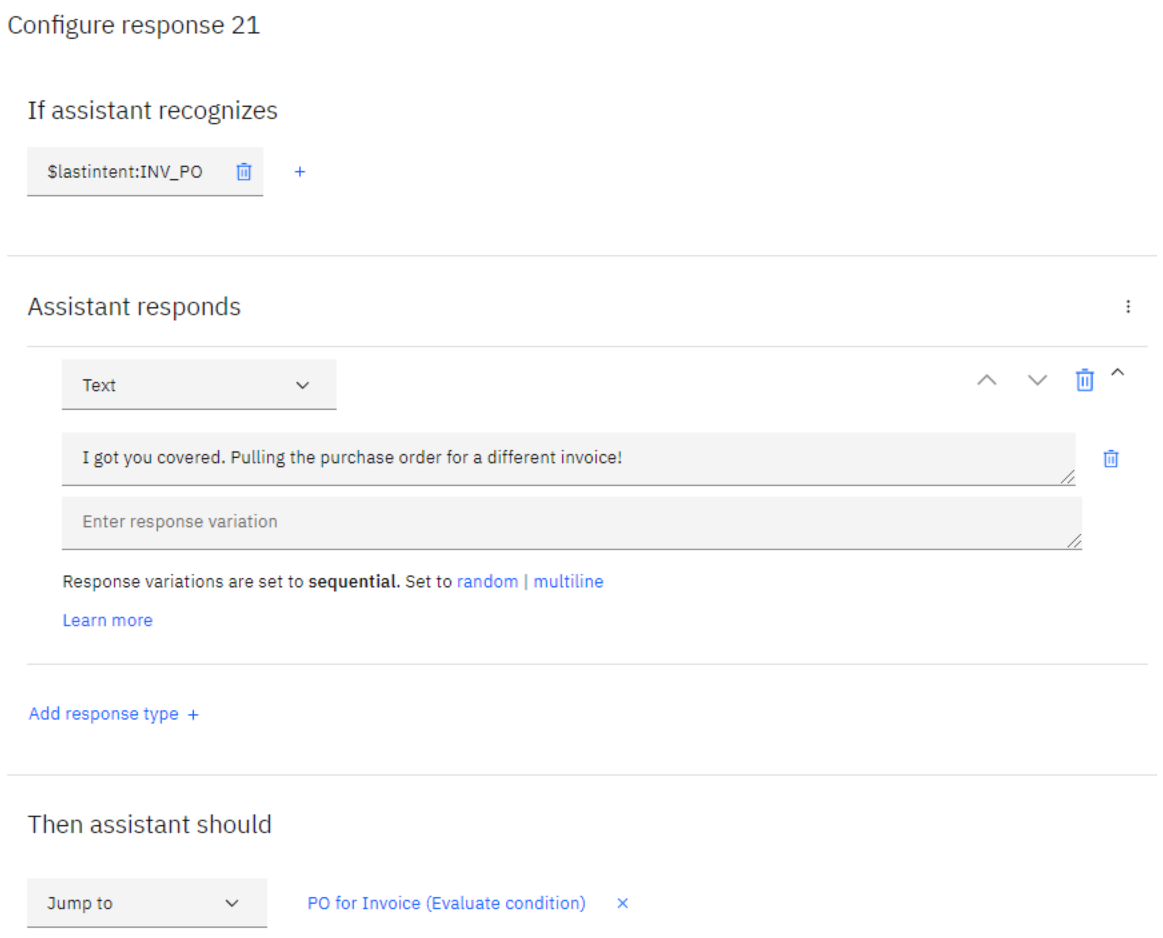
- ORDER_INV - Here, the input is Sales order ID and output is Invoice number.
  Ex: Which Invoice contains sales order xxxxxx ?
- INV_PO - Here, input is Invoice number and output is PO(Purchase Order).
  Ex: I want PO for invoice xxxxxx
- INV_ORDER - Here, the input is invoice number and output is Sales Order number.
  Ex: get me sales order for the invoice xxxxxx
- EPRICER_PO - Here, the input is epricer ID and output is PO(Purchase Order).
  Ex: get me po for epricer ID xxxxxx
- PLNTORDER_INV - Here, the input is plant order and output is Invoice number.
  Ex: what is the invoice for plant order xxxxxx ?
- PLNTORDER_ORDER - Here, the input is plant order and output is Sales Order number.
  Ex: Which sales order contains plant order xxxxxx ?
- PLNTORDER_EPRICER - Here, the input is Plant Order and output is epricer ID.
  Ex: what epricer does xxxxxx plant order belongs to ?

The following entities follow patterns and are trained using Regular Expressions.

- Quotes - 10 digit numbers starting with number 8 but not 83 or 89
- Sales Orders - 10 digit numbers starting with 72 or 73
- Plant Orders - 6 length alphanumerics starting with an alphabet
- Invoices - 10 digit numbers starting with 6 or 83
- Epricers - 7 digit numbers

In the dialogue tree, every intent trained is associated with a node. Each node is configured with slots, context variables and multiple responses as suited. When a node condition is met, it checks for slots. If the slot variables are not present, Watson prompts the user explicitly to provide them. Watson responds accordingly based on whether the slot information is found or not found. Then based on the condition met in the response section, Watson replies accordingly. At this point, a few context variables are set so Watson can remember them through the course of conversation. After responding back to the user, Watson then waits for reply from the user to proceed to the evaluate it again.

A special intent, *repeatPrevInt* is trained so the watson can come back to the same node if asked by the user. It is triggered when a user wants to repeat the previous query with a different entity. The dialogue node is configured in such a way that when the repeatPrevInt Node is processed, watson jumps to the Node which was triggered just before this query. This is possible because when Watson responds to the user, a context variable called *lastintent* is set to the intent recognized.


Configure response 21

If assistant recognizes

$lastintent:INV_PO

Assistant responds

Text

I got you covered. Pulling the purchase order for a different invoice!

Enter response variation

Response variations are set to **sequential.** Set to random | multiline

Learn more

Add response type +

Then assistant should

Jump to

PO for Invoice (Evaluate condition)

Figure 4.1: Response configuration in repeatPrevInt Node

4.2 Creating a web application to deploy the chatbot

The Python web framework, Django is used to create the web application. A simple minimal web page is created with a title bar. A chat circle is place at the bottom right of the web page. On clicking it, a chatbox opens up where the user can chat with the Assistant. The webpage designed is both Desktop and Phone compatible. The messages are generated using JavaScript functions.

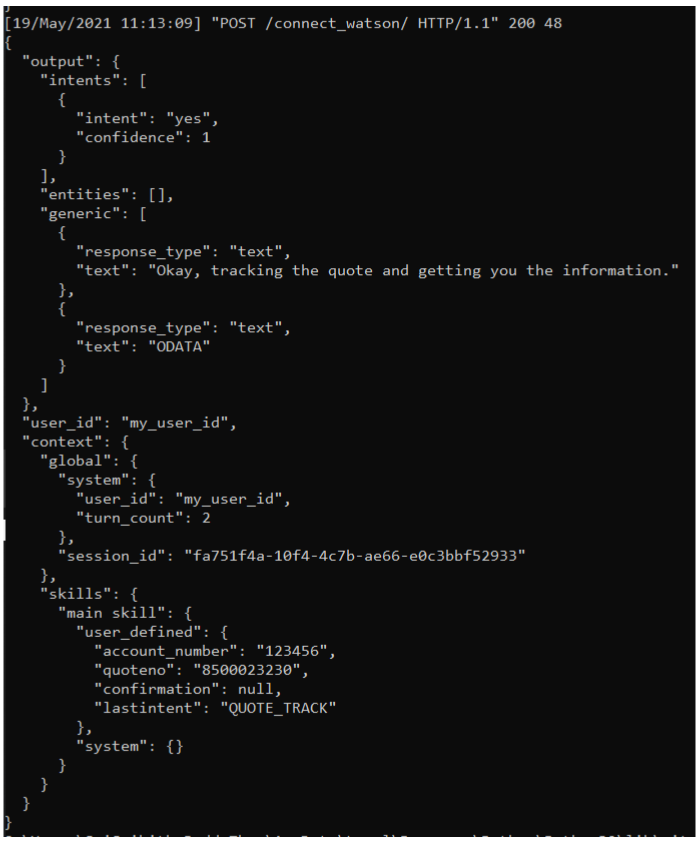
4.3 Connecting web application with Watson Assistant

To connect the Watson Assistant to the backend of the web application, ibm-watson and ibm-cloud-sdk-core python libraries are used.
To send a message to Watson Assistant, authentication is required. The ibm-cloud-sdk-core package supports IBM cloud Identity and Access Management (IAM) authentication for Watson Assistant. The service credentials - Assistant ID and IAM token can be accessed from the API details of the Watson Assistant. A base URL is used as the service URL for Watson. An assistant is initiated with the *AssistantV2* method from ibm-watson package.

To remeber the context variables through the course of conversation with the user, Watson Assistant needs a session. A session is used to send user input to a skill and receive responses. It also maintains the state of the conversation. A session persists until it is deleted, or until it times out because of inactivity. The *create_session* method is used to create a session with the current Assistant instance. When the website is loaded, the process of authenticating to watson and creating the session must be done automatically. An ajax call is made for this purpose with the help of JQuery function *window.onload*.

The textbox where the user enters the input is a HTML form. So the user input is brought to the backend using an ajax call again. The user input is then sent to the Watson Assistant with the *message* method along with assistant ID and watson session ID.

The response received from the Watson Assistant is in the form of a JavaScript Object Notation (JSON) object. A normal dialogue is of type *text*. Figure 4.2 shows an example JSON response received from the Watson Assistant. The JSON response consists of the intent recognized along with its confidence score. It also contains any entities recognized in the user input. The generic field of the JSON object contains the responses along with their type. Here, the responses are of type text. The context field contains the session ID which was passed with the message method. The skills section consists of all the context variables set when the user input was evaluated.



```
[19/May/2021 11:13:09] "POST /connect_watson/ HTTP/1.1" 200 48
{
  "output": {
    "intents": [
      {
        "intent": "yes",
        "confidence": 1
      }
    ],
    "entities": [],
    "generic": [
      {
        "response_type": "text",
        "text": "Okay, tracking the quote and getting you the information."
      },
      {
        "response_type": "text",
        "text": "ODATA"
      }
    ]
  },
  "user_id": "my_user_id",
  "context": {
    "global": {
      "system": {
        "user_id": "my_user_id",
        "turn_count": 2
      },
      "session_id": "fa751f4a-10f4-4c7b-ae66-e0c3bbf52933"
    },
    "skills": {
      "main skill": {
        "user_defined": {
          "account_number": "123456",
          "quoteno": "8500023230",
          "confirmation": null,
          "lastintent": "QUOTE_TRACK"
        },
        "system": {}
      }
    }
  }
}
```

Figure 4.2: Example JSON response from Watson Assistant

The text responses are all collected in the form of a text array and returned to the ajax call so the messages can be generated for the user in the chatbox. When a session time out occurs while receiving the response from Watson Assistant, an ApiException is returned from the message method instead of the usual JSON object. This ApiException is caught in the backend and the ajax call will receive the response of *Invalid Session*. When this happens, it is known that session time out has occured and that a new watson instance needs to be created.

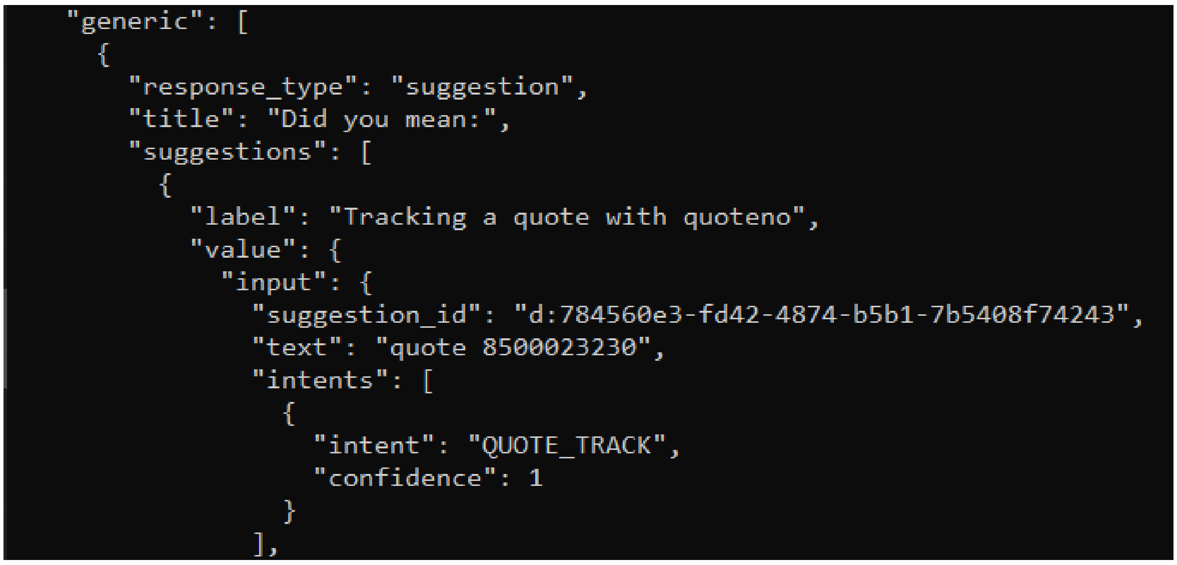
4.4 Connecting web aplication to the database

OData is a REST based protocol for querying and updating data. Based on the intent and entities gathered in Section 4.3, the OData API URL is appended with a suitable filter to query the data from the database. A *requests.get* call is made to the database for the data. The Accept field in headers parameter of the request method is set to application/json so the response received is a JSON object.
The flow is dependent on the intent recognized by the Watson Assistant. The JSON

object received contains all the values under results field. If the results field is empty, it is known that there were no records matching the filter in the database. The values gathered from this JSON response are parsed into a neat English sentence and sent back to the client UI chatbox for the user.

4.5 Toggling Disambiguation and Autolearning

Disambiguation and Autolearning are enabled for the Assistant to make it more genuine and smarter. Watson SDK responds with a suggestion type response for disambiguations in the JSON object.



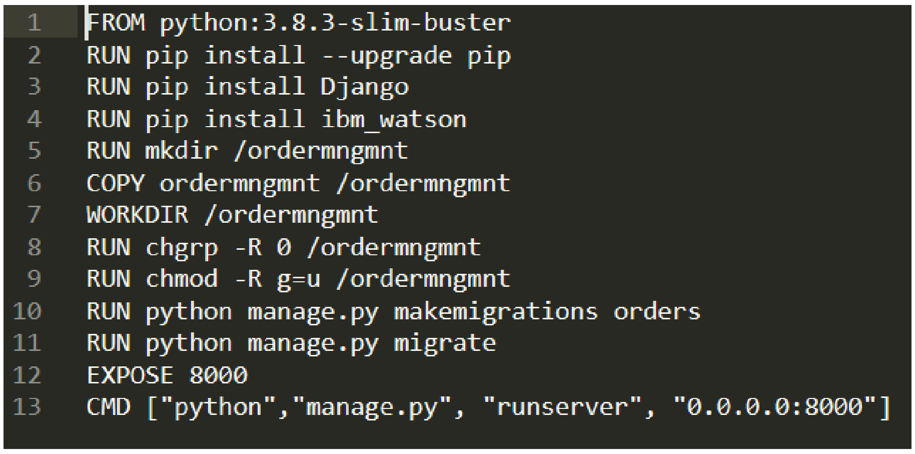
```
  "generic": [
    {
      "response_type": "suggestion",
      "title": "Did you mean:",
      "suggestions": [
        {
          "label": "Tracking a quote with quoteno",
          "value": {
            "input": {
              "suggestion_id": "d:784560e3-fd42-4874-b5b1-7b5408f74243",
              "text": "quote 8500023230",
              "intents": [
                {
                  "intent": "QUOTE_TRACK",
                  "confidence": 1
                }
              ],
```

Figure 4.3: Example JSON response for Disambiguations

Figure 4.3 shows the response type to be suggestion when the disambiguation is set off. The suggestions field contains an array of suggestions made by Watson for the user to choose from. The label field is the description of a dialogue Node. When suggestions are made, the user is prompted to enter an option to proceed. Any response to the suggestions other than the number associated with the option is not accepted. Watson will keep at it until an acceptable integer option is entered. When an appropriate option is received from the user, the respective dictionary value of the input field is passed as a parameter to the message method instead of the regular user query. The flow is brought back to normal as Watson responds back with a JSON object again.

4.6 Dockerizing the web application

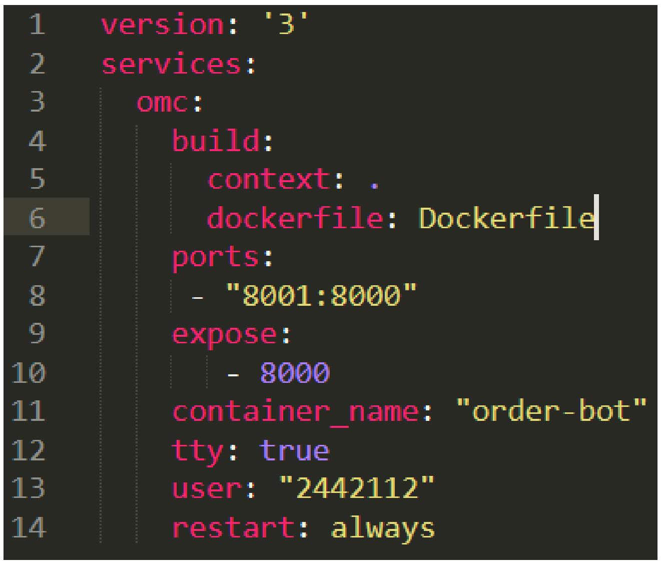
To deploy the web application onto Cirrus, IBM's private cloud, it needs to be dockerized. A dockerfile and a docker-compose yaml file are written so docker can build an image.



```
FROM python:3.8.3-slim-buster
RUN pip install --upgrade pip
RUN pip install Django
RUN pip install ibm_watson
RUN mkdir /ordermngmnt
COPY ordermngmnt /ordermngmnt
WORKDIR /ordermngmnt
RUN chgrp -R 0 /ordermngmnt
RUN chmod -R g=u /ordermngmnt
RUN python manage.py makemigrations orders
RUN python manage.py migrate
EXPOSE 8000
CMD ["python","manage.py", "runserver", "0.0.0.0:8000"]
```

Figure 4.4: Dockerfile

In the dockerfile (Figure 4.4), the base image is python slim buster. Django, ibm_watson packages are installed on top of it. The code folders are then copied. Appropriate permissions to read, view and excute the folders are given. The makemigrations and migrate commands are for Django sessions. Port 8000 is exposed for documentation purpose. The last line runs the web application from the command line.



```
version: '3'
services:
  omc:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
     - "8001:8000"
    expose:
      - 8000
    container_name: "order-bot"
    tty: true
    user: "2442112"
    restart: always
```

Figure 4.5: Docker compose file

In the docker-compose file (Figure 4.5), restart:always command restarts containers when they exit with zero exit code.

4.7 Deploying the container onto Cirrus

Once the docker image is ready, a project is created on the Cirrus. The project created is internet facing so it can be accessed over the internet with a URL. Each project can have multiple pipelines and applications, each with their own configurations. A pipeline is created and the docker image is uploaded to it. Multiple image versions can be uploaded to a pipeline which are differentiated by the tag name as shown in Figure 4.6.

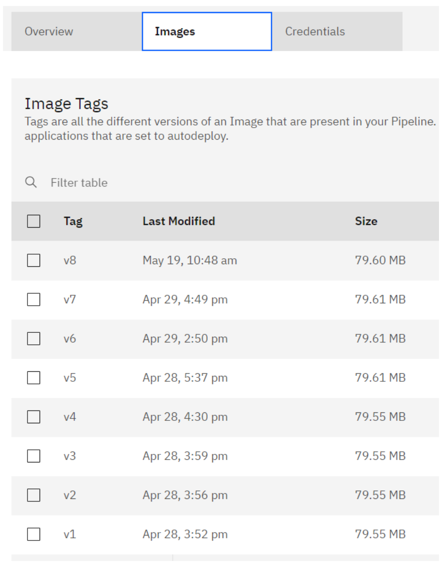
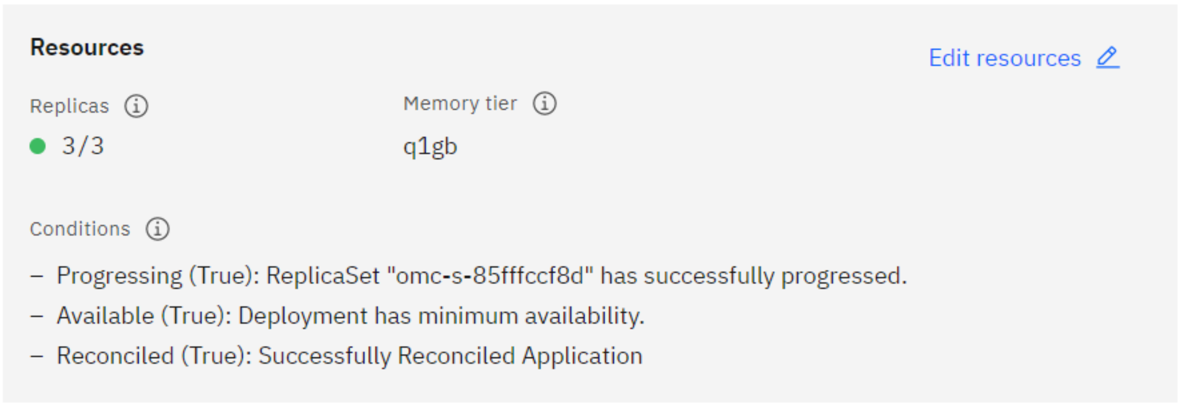


Image Tags
Tags are all the different versions of an Image that are present in your Pipeline. applications that are set to autodeploy.

☐	Tag	Last Modified	Size
☐	v8	May 19, 10:48 am	79.60 MB
☐	v7	Apr 29, 4:49 pm	79.61 MB
☐	v6	Apr 29, 2:50 pm	79.61 MB
☐	v5	Apr 28, 5:37 pm	79.61 MB
☐	v4	Apr 28, 4:30 pm	79.55 MB
☐	v3	Apr 28, 3:59 pm	79.55 MB
☐	v2	Apr 28, 3:56 pm	79.55 MB
☐	v1	Apr 28, 3:52 pm	79.55 MB

Figure 4.6: Various images with different tag names uploaded to a pipeline

An application is then created with the pipeline. Each application is allotted resources (Figure 4.7). Here, 3 pods and a memory of 1GB is allotted to the application.



Resources

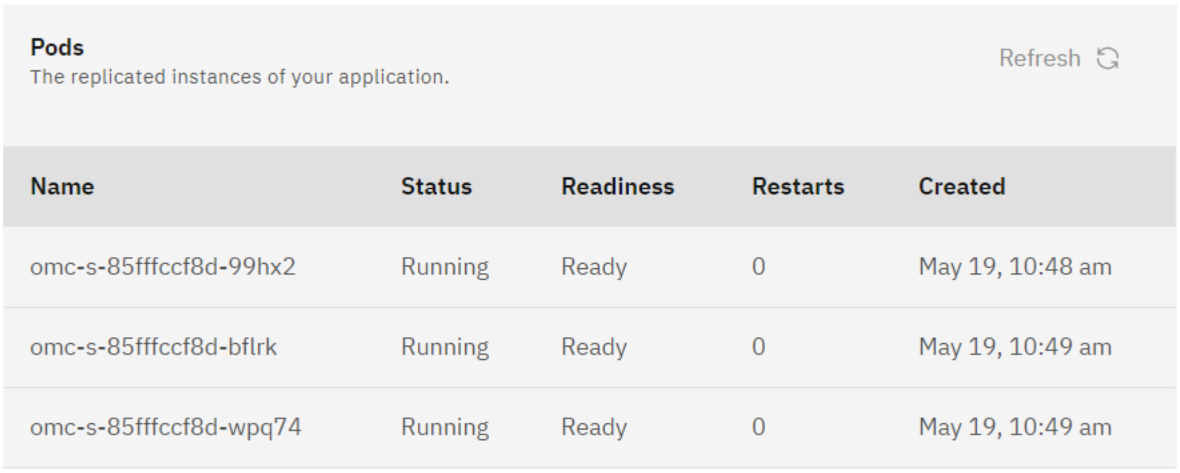
Replicas	Memory tier
● 3/3	q1gb

Conditions

- Progressing (True): ReplicaSet "omc-s-85fffccf8d" has successfully progressed.
- Available (True): Deployment has minimum availability.
- Reconciled (True): Successfully Reconciled Application

Figure 4.7: Resources for a Cirrus Application

Pods in Cirrus are replica instances of the application which run the same image. This provides availability and keeps the web application accessible to users in case one or the other instances is down.


Pods
The replicated instances of your application.

Refresh

Name	Status	Readiness	Restarts	Created
omc-s-85fffccf8d-99hx2	Running	Ready	0	May 19, 10:48 am
omc-s-85fffccf8d-bflrk	Running	Ready	0	May 19, 10:49 am
omc-s-85fffccf8d-wpq74	Running	Ready	0	May 19, 10:49 am

Figure 4.8: Pods allotted to a Cirrus Application

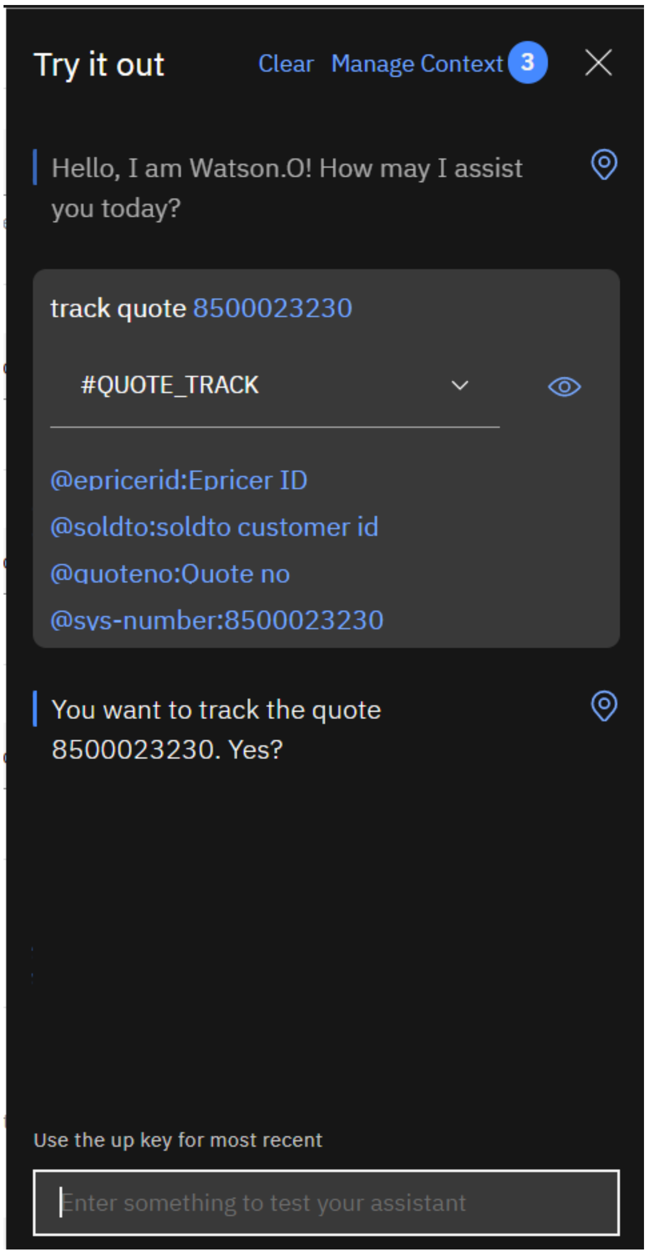
By default, the firewall in Cirrus blocks any outgoing or incoming requests to the application. To allow the OData requests to go through the application to the API layer, the respective Destination IP address and Port are added to the firewall list.

4.8 Integrating the Cirrus application with LogDNA

For security purposes it is important to store the logs. They can be used to point out a red flag and help mitigate the issue. Cirrus allows the projects to be integrated with LogDNA, a log management solution directly.

4.9 Testing the Watson Assistant

Not all humans are same and hence their way of chatting would also be quite different and unique. To make Watson understand all kinds of Natural Language queries, it is tested thoroughly in the try it out panel (Figure 4.9) of the dialogue skill. The changes made to the Assistant are reflected directly in this panel. By querying the Watson in different ways, the confidence scores of the top recognized intent is checked and made sure that it stays above 0.2 so the user can receive a suitable response.


Try it out

Clear Manage Context 3

Hello, I am Watson.O! How may I assist you today?

track quote 8500023230

#QUOTE_TRACK

@epricerid:Epricer ID
@soldto:soldto customer id
@quoteno:Quote no
@sys-number:8500023230

You want to track the quote 8500023230. Yes?

Use the up key for most recent

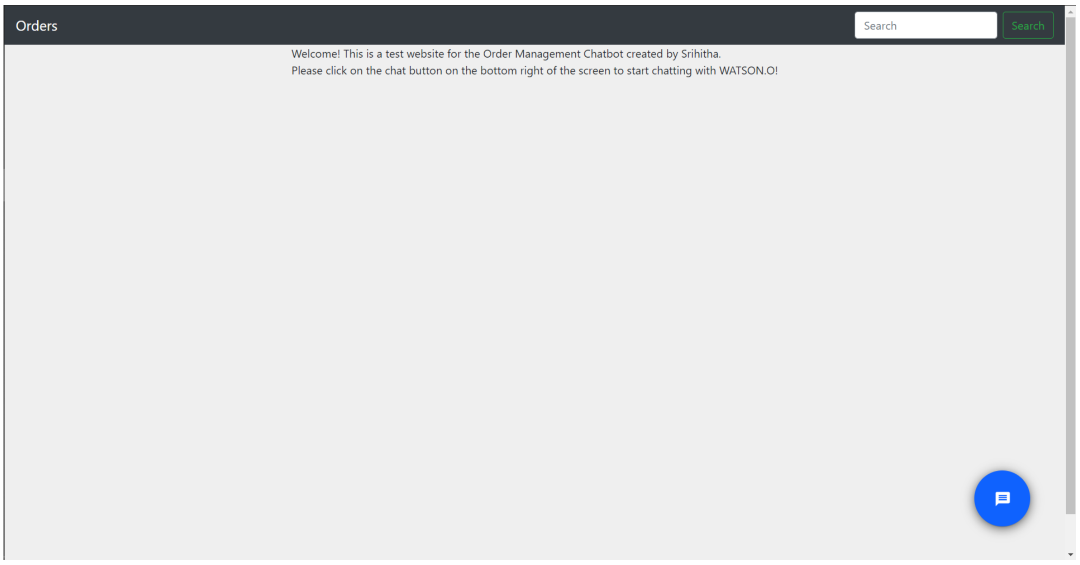
Enter something to test your assistant

Figure 4.9: Try it out panel

All the intents are made more confident by adding appropriate examples. The intent testing is also done from the Client end to make sure the queries are fetching apt data for each intent from the database. Digression feature is also tested to check how the Assitant responds when the user is trying to deviate from the conversation.

Chapter 5

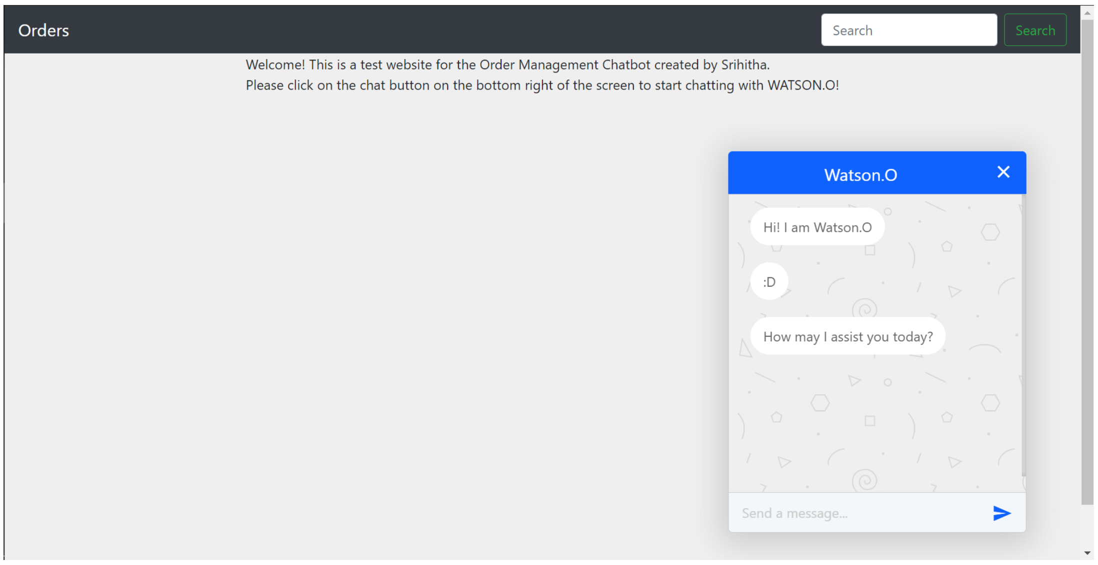
RESULTS AND DISCUSSION



Orders

Welcome! This is a test website for the Order Management Chatbot created by Srihitha.
Please click on the chat button on the bottom right of the screen to start chatting with WATSON.O!

Figure 5.1: Client UI



Orders

Welcome! This is a test website for the Order Management Chatbot created by Srihitha.
Please click on the chat button on the bottom right of the screen to start chatting with WATSON.O!

Watson.O

Hi! I am Watson.O

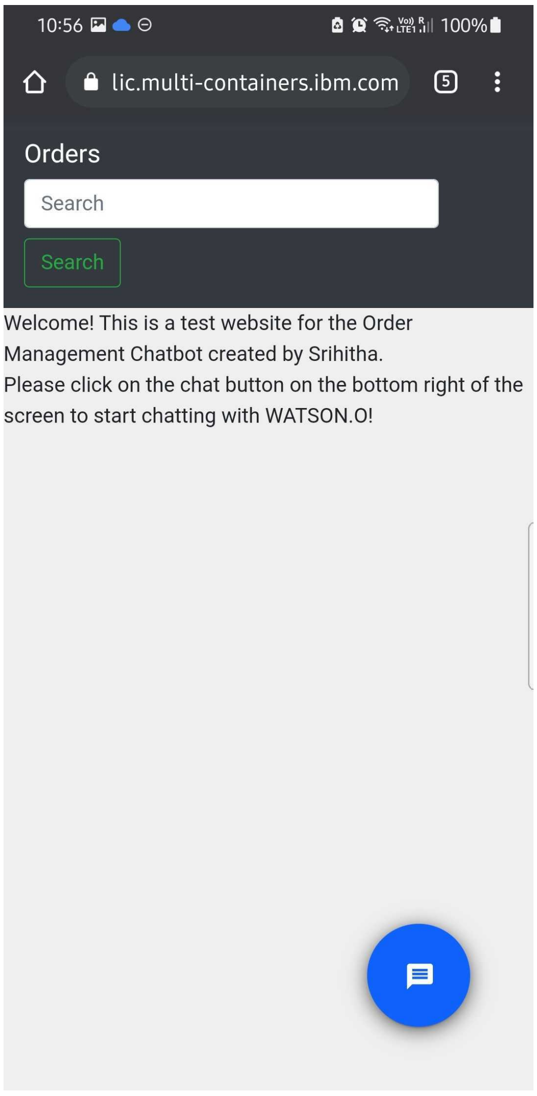
:D

How may I assist you today?

Figure 5.2: Chatbox which opens after clicking on the chat icon

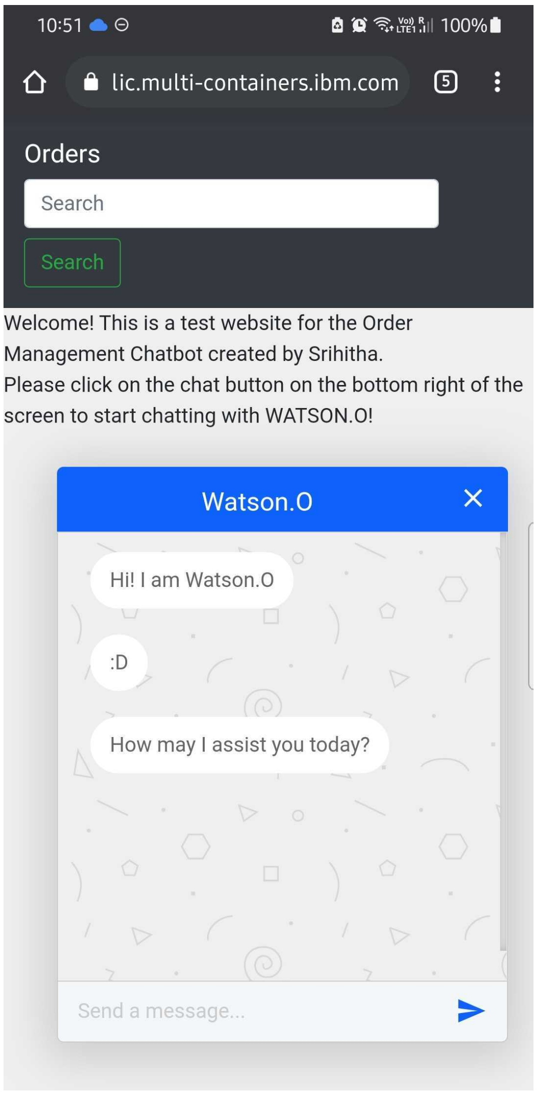
The above images show how the website looks on Desktop and Personal Computers. On clicking the chat icon shown in Figure 5.1, a chatbox pops up and Watson greets the user. This is when a Watson Session is automatically started, as explained in Section

4.3. The chatbox, as shown in Figure 5.2, has a text input area where the user can enter the query.



Welcome! This is a test website for the Order Management Chatbot created by Srihitha.
Please click on the chat button on the bottom right of the screen to start chatting with WATSON.O!

(a) UI with Chat icon



Welcome! This is a test website for the Order Management Chatbot created by Srihitha.
Please click on the chat button on the bottom right of the screen to start chatting with WATSON.O!

Hi! I am Watson.O

:D

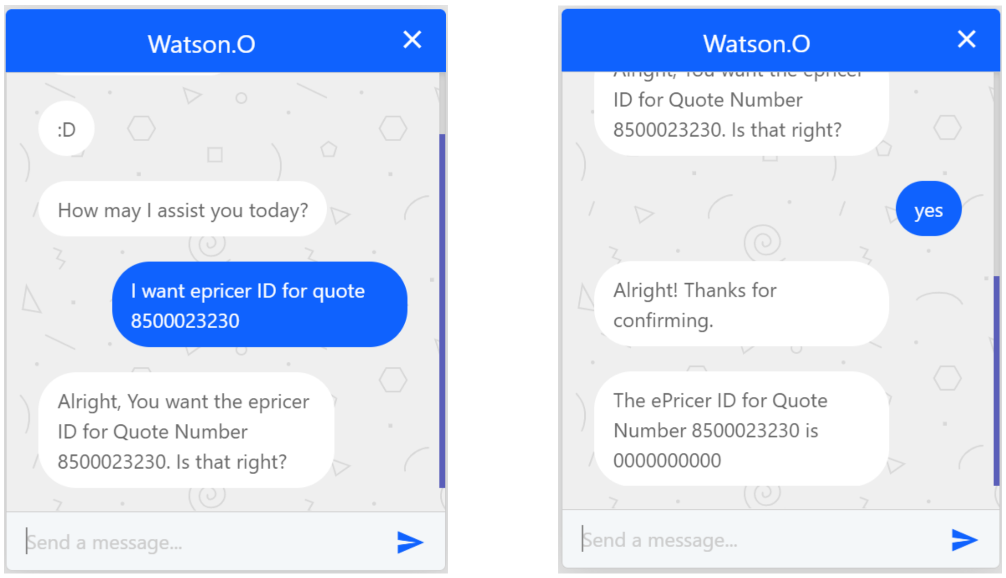
How may I assist you today?

(b) UI with Chatbox

Figure 5.3: Client UI on Phone

The above images show how the website looks on Phone.

When a user enters a query with input entity, Watson asks for confirmation from the user if the query and entity recognized are correct. Once that confirmation is provided by the user, the respective data is queried with the OData API and parsed into a normal sentence for the user.


Watson.O

:D

How may I assist you today?

I want epricer ID for quote 8500023230

Alright, You want the epricer ID for Quote Number 8500023230. Is that right?

Send a message...

Watson.O

ID for Quote Number 8500023230. Is that right?

yes

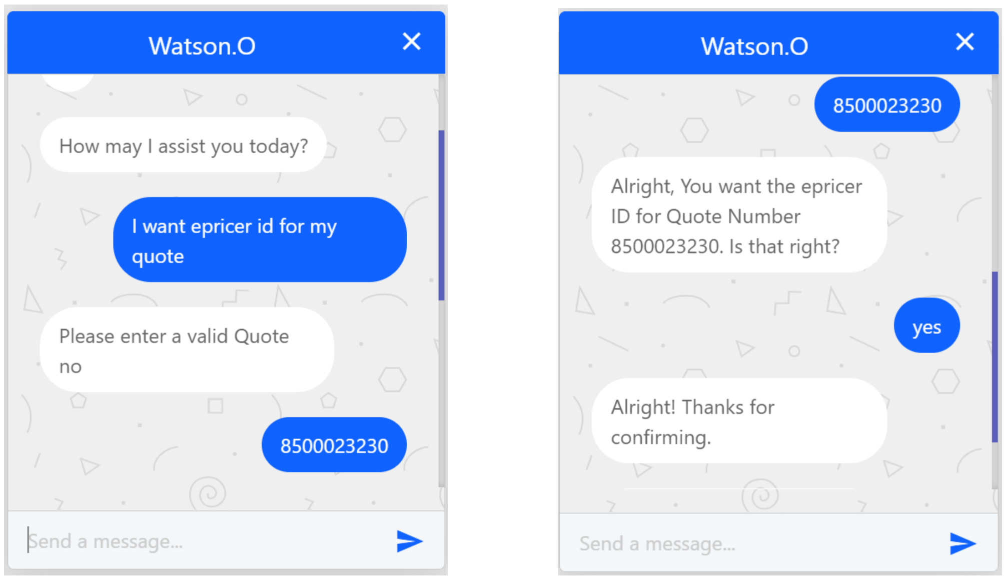
Alright! Thanks for confirming.

The ePricer ID for Quote Number 8500023230 is 0000000000

Send a message...

Figure 5.4: A normal dialogue exchange between user and Watson

If the user does not provide the required entity, slots are activated and the user is prompted for them explicitly. Once the information is provided, the dialogue flow goes back to normal and Watson asks for confirmation from the user regarding the query.


Watson.O

How may I assist you today?

I want epricer id for my quote

Please enter a valid Quote no

8500023230

Send a message...

Watson.O

8500023230

Alright, You want the epricer ID for Quote Number 8500023230. Is that right?

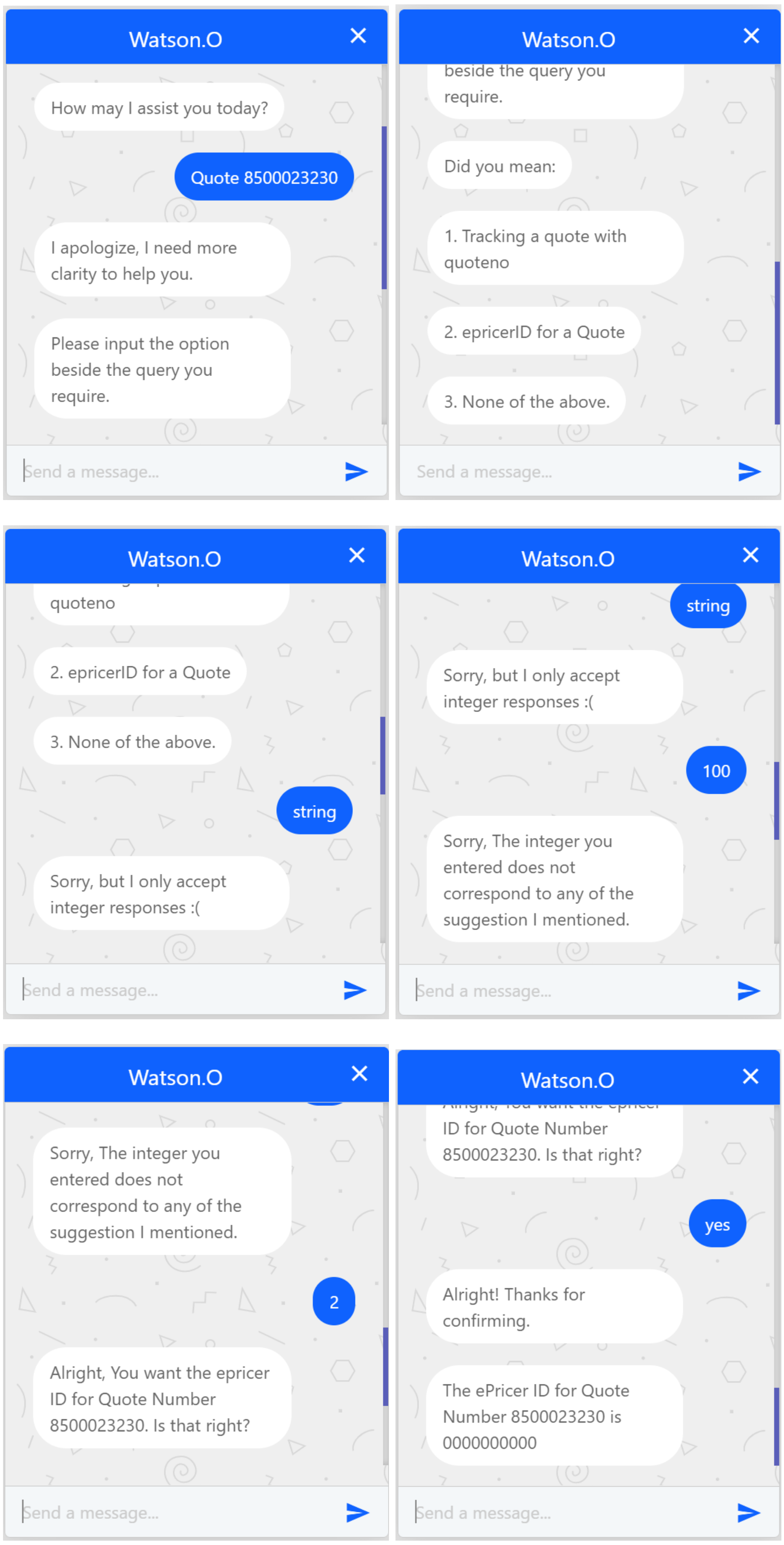
yes

Alright! Thanks for confirming.

Send a message...

Figure 5.5: Watson prompting the user for entity

Disambiguation is triggered when Watson is confident that two or more recognized intents can answer the user's query. In this case, a menu is provided to the user to choose from. Only valid integer responses are accepted by Watson when this situation arises.



How may I assist you today?

Quote 8500023230

I apologize, I need more clarity to help you.

Please input the option beside the query you require.

beside the query you require.

Did you mean:

1. Tracking a quote with quoteno

2. epricerID for a Quote

3. None of the above.

quoteno

2. epricerID for a Quote

3. None of the above.

string

Sorry, but I only accept integer responses :(

string

Sorry, but I only accept integer responses :(

100

Sorry, The integer you entered does not correspond to any of the suggestion I mentioned.

Sorry, The integer you entered does not correspond to any of the suggestion I mentioned.

2

Alright, You want the epricer ID for Quote Number 8500023230. Is that right?

ID for Quote Number 8500023230. Is that right?

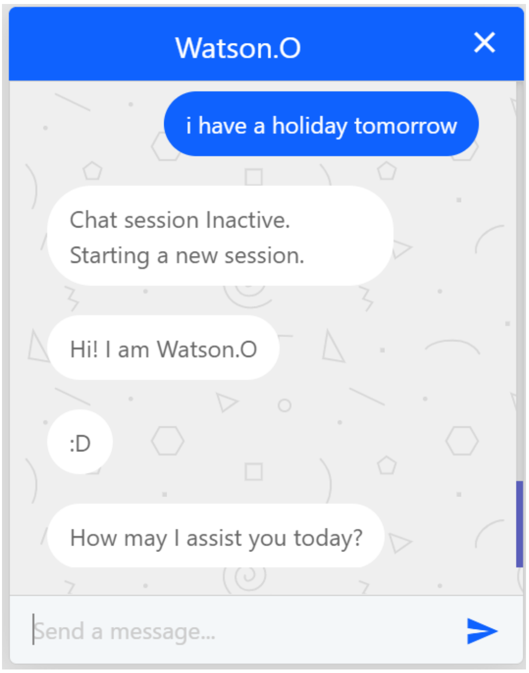
yes

Alright! Thanks for confirming.

The ePricer ID for Quote Number 8500023230 is 0000000000

Figure 5.6: Watson responding to a Disambiguation scenario

When a session time out is encountered, Watson lets the user know about it and greets the user anew, indicating that a new session has been started.


Watson.O

i have a holiday tomorrow

Chat session Inactive.
Starting a new session.

Hi! I am Watson.O

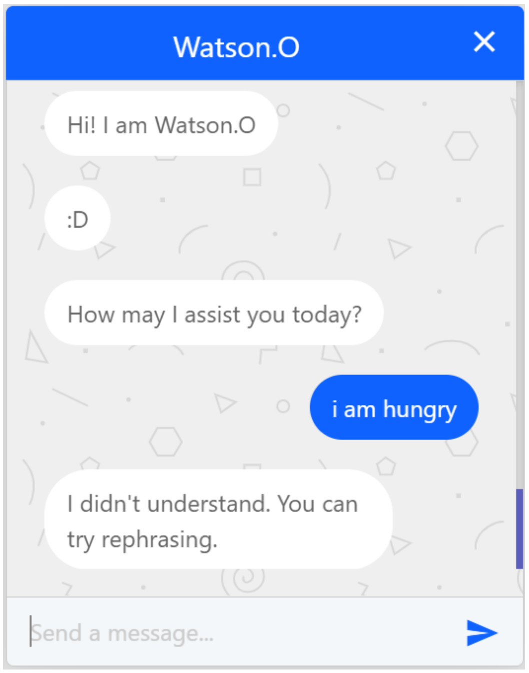
:D

How may I assist you today?

Send a message...

Figure 5.7: Session Time Out scenario

Watson is also capable of detecting and replying to irrelevant queries.


Watson.O

Hi! I am Watson.O

:D

How may I assist you today?

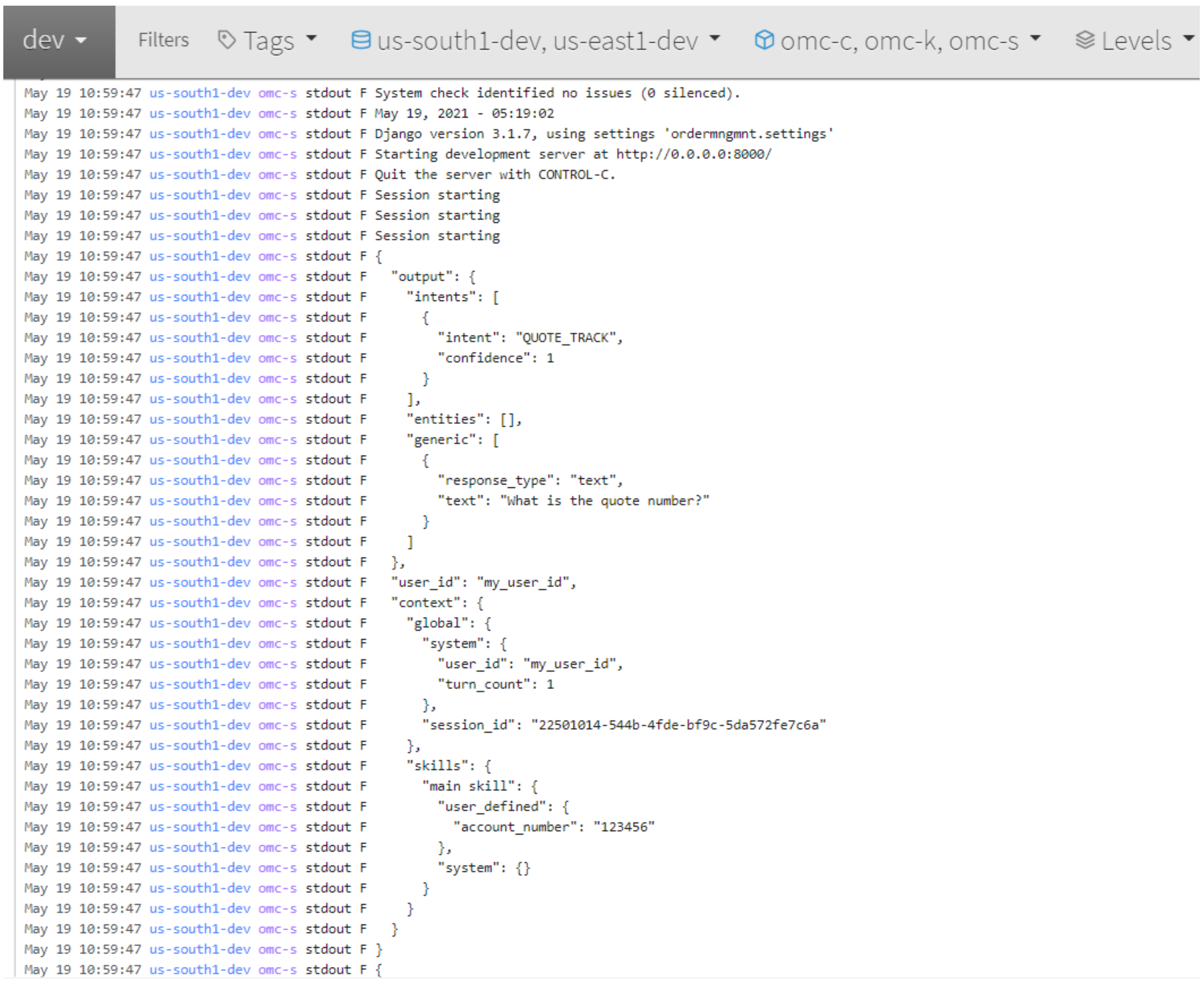
i am hungry

I didn't understand. You can try rephrasing.

Send a message...

Figure 5.8: Irrelevant query scenario

After integrating LogDNA with the Cirrus application, logs can be viewed.



```
May 19 10:59:47 us-south1-dev omc-s stdout F System check identified no issues (0 silenced).
May 19 10:59:47 us-south1-dev omc-s stdout F May 19, 2021 - 05:19:02
May 19 10:59:47 us-south1-dev omc-s stdout F Django version 3.1.7, using settings 'ordermngmnt.settings'
May 19 10:59:47 us-south1-dev omc-s stdout F Starting development server at http://0.0.0.0:8000/
May 19 10:59:47 us-south1-dev omc-s stdout F Quit the server with CONTROL-C.
May 19 10:59:47 us-south1-dev omc-s stdout F Session starting
May 19 10:59:47 us-south1-dev omc-s stdout F Session starting
May 19 10:59:47 us-south1-dev omc-s stdout F Session starting
May 19 10:59:47 us-south1-dev omc-s stdout F {
May 19 10:59:47 us-south1-dev omc-s stdout F   "output": {
May 19 10:59:47 us-south1-dev omc-s stdout F     "intents": [
May 19 10:59:47 us-south1-dev omc-s stdout F       {
May 19 10:59:47 us-south1-dev omc-s stdout F         "intent": "QUOTE_TRACK",
May 19 10:59:47 us-south1-dev omc-s stdout F         "confidence": 1
May 19 10:59:47 us-south1-dev omc-s stdout F       }
May 19 10:59:47 us-south1-dev omc-s stdout F     ],
May 19 10:59:47 us-south1-dev omc-s stdout F     "entities": [],
May 19 10:59:47 us-south1-dev omc-s stdout F     "generic": [
May 19 10:59:47 us-south1-dev omc-s stdout F       {
May 19 10:59:47 us-south1-dev omc-s stdout F         "response_type": "text",
May 19 10:59:47 us-south1-dev omc-s stdout F         "text": "What is the quote number?"
May 19 10:59:47 us-south1-dev omc-s stdout F       }
May 19 10:59:47 us-south1-dev omc-s stdout F     ]
May 19 10:59:47 us-south1-dev omc-s stdout F   },
May 19 10:59:47 us-south1-dev omc-s stdout F   "user_id": "my_user_id",
May 19 10:59:47 us-south1-dev omc-s stdout F   "context": {
May 19 10:59:47 us-south1-dev omc-s stdout F     "global": {
May 19 10:59:47 us-south1-dev omc-s stdout F       "system": {
May 19 10:59:47 us-south1-dev omc-s stdout F         "user_id": "my_user_id",
May 19 10:59:47 us-south1-dev omc-s stdout F         "turn_count": 1
May 19 10:59:47 us-south1-dev omc-s stdout F       },
May 19 10:59:47 us-south1-dev omc-s stdout F       "session_id": "22501014-544b-4fde-bf9c-5da572fe7c6a"
May 19 10:59:47 us-south1-dev omc-s stdout F     },
May 19 10:59:47 us-south1-dev omc-s stdout F     "skills": {
May 19 10:59:47 us-south1-dev omc-s stdout F       "main skill": {
May 19 10:59:47 us-south1-dev omc-s stdout F         "user_defined": {
May 19 10:59:47 us-south1-dev omc-s stdout F           "account_number": "123456"
May 19 10:59:47 us-south1-dev omc-s stdout F         },
May 19 10:59:47 us-south1-dev omc-s stdout F         "system": {}
May 19 10:59:47 us-south1-dev omc-s stdout F       }
May 19 10:59:47 us-south1-dev omc-s stdout F     }
May 19 10:59:47 us-south1-dev omc-s stdout F   }
May 19 10:59:47 us-south1-dev omc-s stdout F }
May 19 10:59:47 us-south1-dev omc-s stdout F {
```

Figure 5.9: Application logs in LogDNA

Chapter 6

CONCLUSION

In this project, a chatbot is created using Watson Assistant Service provided by IBM. The Assistant is properly trained with suitable intents and entites. A dialogue tree is constructed so the Assistant can respond to the user efficiently and effectively utlising various features such as slots, context variables and mutliple responses. This assistant is then deployed onto a web application created with Django. The user input is sent to Watson Assistant using ibm_watson package. The response received is in the form of a JSON object where the intent and entities recognized are gathered. Based on the findings, OData URL is appended with an apt filter to query the data from the database. The data received, again in a JSON object, is then parsed into a user understandable sentence. This response is delivered to the user in the chatbox. For enhanced communication, Autolearning and Disambiguation features are enabled for the Assistant. The whole web application is dockerized and deployed onto Cirrus. The Cirrus Application is integrated with LogDNA to store the logs in one place. Watson Assistant is also tested to make sure it provides a relevant response to the user.

REFERENCES

1. Bootstrap, T. “Browsers and devices”. https://getbootstrap.com/docs/4.5/getting-started/browsers-devices/. Accessed: 2021-05-24.

2. Botscrew. “5 industries that benefit from chatbots already”. https://botscrew.com/blog/industries-benefit-chatbots/. Accessed: 2021-05-24.

3. Documentation, IBM Cloud. (2020). “Understanding dialog skill digressions”. https://cloud.ibm.com/docs/assistant?topic=assistant-tutorial-digressions/. Accessed: 2021-05-24.

4. Documentation, IBM Cloud. (2021a). “Controlling the conversational flow”. https://cloud.ibm.com/docs/assistant?topic=assistant-dialog-runtime. Accessed: 2021-05-24.

5. Documentation, IBM Cloud. (2021b). “Creating a Dialog”. https://cloud.ibm.com/docs/assistant?topic=assistant-dialog-overview. Accessed: 2021-05-24.

6. Documentation, IBM Cloud. (2021c). “Enable your skill to improve itself with autolearning”. https://cloud.ibm.com/docs/assistant?topic=assistant-autolearn. Accessed: 2021-05-24.

7. Documentation, IBM Cloud. (2021d). “Personalizing the dialog with context”. https://cloud.ibm.com/docs/assistant?topic=assistant-dialog-runtime-context. Accessed: 2021-05-24.

8. Feng, D. “The World of Chatbots: Customer Service, Business Automation and Scalability”. https://www.bigcommerce.com/blog/chatbots/. Accessed: 2021-05-24.

9. Greyling, C. “Your Chatbot Must Be Able To Disambiguate”. https://cobusgreyling.me/your-chatbot-must-be-able-to-disambiguate/. Accessed: 2021-05-24.

10. Team, E. (2020). “Chatbot: What is a Chatbot? Why are Chatbots Important?”. https://www.expert.ai/blog/chatbot/. Accessed: 2021-05-24.