“Generic Chatbot Augmentation Platform”

An Internship Report

Submitted By:

Manik Khurana

R110216092

500051975

Program: B.Tech (CSE-CCVT), 7th Semester

Department of Virtualization

To

Mr. Prakash GL

Assistant Professor (SG)

Department of Virtualization

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

University of Petroleum and Energy Studies

Bidholi, Via Prem Nagar

Dehradun, Uttarakhand - 248007

ACKNOWLEDGEMENT

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report. A special gratitude we give to our semester technical report teacher, Mr.Prakash GL, whose contribution in stimulating suggestions and encouragement, helped us to coordinate our technical report especially in writing this report.

Furthermore, we would also like to acknowledge with much appreciation the crucial role of the staff of UPES, who gave the permission to use all required equipment and the necessary materials to complete the report on "Generic Chatbot Augmentation Platform". Last but not least, many thanks go to IEEE, their references, and other technical reports. I have to appreciate the guidance given by another supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advice.

TO WHOMSOEVER IT MAY CONCERN

This is to certify that this project report entitled **"Generic Chatbot Augmentation Platform"** submitted to UPES is a bonafide record of work done by "Manik Khurana, 3rd year student of B.Tech Computer Science and Engineering spl. in Cloud Computing and Virtualization Technology, UPES" under my supervision.

This project is the record of authentic work carried out during the academic year 2019-20.

In the span of this technical report their candidature were found to be very sincere & hardworking.

Signature

Mr. Prakash GL
Assistant Professor (SG)
Department of Virtualization

TABLE OF CONTENTS

LIST OF FIGURES

1. ABSTRACT

The Project titled “Generic Chatbot Augmentation Platform” is a tool for a user where they can request a chatbot for their website. The Platform will be outputting a simple link that can be embedded in any website’s frontend. The chatbot can be powered by any service provider, Google’s Dialogflow, Amazon Lex, Microsoft LUIS or IBM Watson. The Generic nature of the software in question allows the developer/administrator to simply attach-detach the provider from behind. In other words, the client can request for any service provider or their plans and according to the budget & allowances the administrator can route their requests to that particular provider. This enables a simple add/remove package strategy to the software where module by module development is possible and integration simplicity is maxed out. The project as visible to the user will just be a portal dashboard where he/she, as a super-admin, control the admins. And these admins in command can control the data that is being fed to the chatbot. The user on the other hand will just be looking at the chatbot and asking it various sorts of questions and this chatbot will be replying to the user accordingly. The Machine Learning technique involved in this project is totally covered by Google’s Dialogflow and the other providers individually. This project deals with remote login and API Calls to the provider servers. The backend of the project is in Node.js, the frontend in Angular.js/HTML/CSS, the ORM used is Sequelize, and the database is MySQL. The generic nature allows us to change the database and the provider without touching the app-level code.

2. INTRODUCTION

The project “Generic Chatbot Augmentation Platform” built at Perennial Systems, Pune during our summer internship period (May’19-Aug’19) is an efficient tool for managing, powering and monitoring various chatbots and their providers. This Augmented Platform enables the client to use any chatbot over the same skin provided by any service provider on a particular webpage. The nature of the project is said to be generic as the app level code is not to be changed even when we want to change the chatbot service provider or the front website for that matter.

The basic inconvenience incurred by users when any issue is not resolved over a phone call or a mail to any organization will now be covered as the chatbot is autonomous and it learns with what you program it for. For example, a chatbot programmed for taking pizza orders will only take pizza orders and ask relevant questions like toppings, sauces, cheese levels, etc. This way a user can order any pizza over a few simple texts.

The users, using this chatbot, can now lodge complaints and do so much more with it. It is a great tool that automates so many different processes and encapsulates them in a single NPM Package for each service Provider.

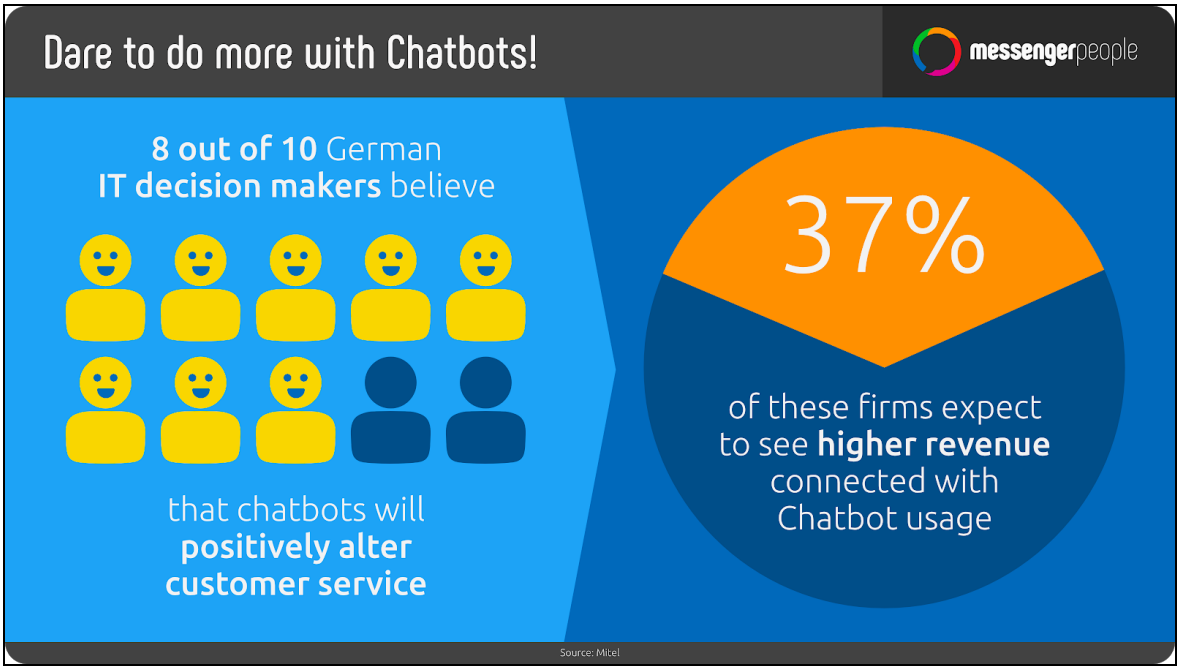

Figure 1: Chatbots rule

Where is the product best in use?

Large companies such as United Airlines, Pizza Hut, Denny's Diner, Focus Features, and Patron, just to name a few, are way ahead in the customer service department. Why? Because they all have chatbots implemented into their social media pages. Have you ever had a problem with a service or a product and ended up contacting the company? Many of us have and chances are that the experience wasn't pleasant. The process can be very frustrating and in most cases, those customer support agents aren't trained very well and have limited knowledge and resources to actually help solve your issue. It's time to move on from those pesky agents! It's 2019 and technology is advancing. In this case, chatbots have started to take on real human support agents. It is way more convenient and way less time consuming to chat with a chatbot. You can have a smooth conversation with it and not have any interruptions because it uses Artificial Intelligence (AI) to find the appropriate response for you. Various apps like Facebook Messenger or Direct Messages on Twitter greatly help since they have the ability to implement chatbots for businesses and brands. Some chatbots are even able to learn from the information that is sent. For example, if a customer wants to order a particular food item, the bot will remember what that person ordered and will ask if he'd like to order it again the next time he returns. Chatbots have the ability to quickly search through large amounts of information to correctly choose the best answers for the customer.

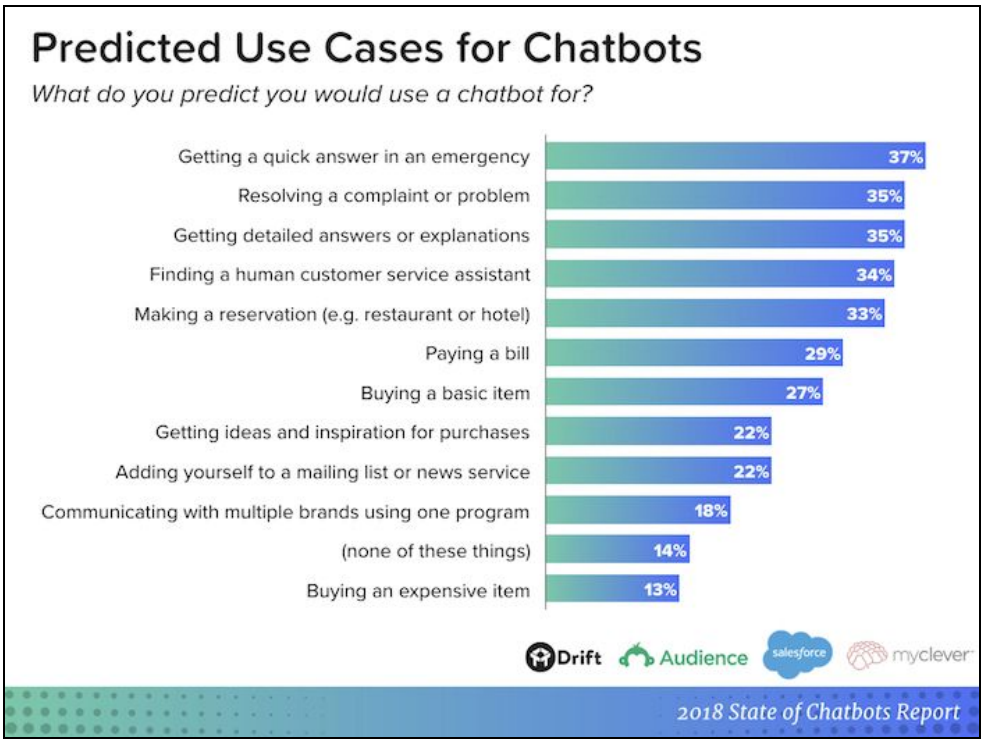

Figure 2: The Predicted use cases of Chatbots

Chatbots can also give us the option to speak to an agent directly. This means that it has the capability of transferring you to a real human support agent. With today's technology and AI/ML, the possibilities are endless.

Chatbots are also very easy to use. With Twitter DMs and Facebook Messenger, you can create bots with Quick Replies. Quick Replies are pre-filled options that a user can choose if one matches their need. For example, a customer is greeted with an automatic welcome message asking them to choose the appropriate option. If they need to find the closest store location, they would select the appropriate Quick Reply. The bot would respond asking them to send their current location and then it would provide them with the store closest to the sent location.

Which Chatbot platforms are easiest to build and train on?

Depending on one's experience with machine learning, programming, and chatbots, the answer will vary. Equipped with some knowledge of machine learning and natural language processing (NLP). this will be a challenging way to find an answer to our question: Can someone that has little to no hands-on experience with conversational agents, programming, or machine learning to build a chatbot? Truth be told, we aren't building a chatbot or machine learning algorithm from scratch. Instead, we are using the framework and algorithm that each chatbot service provider uses. Even though we aren't building a chatbot, we still need to be comfortable with learning new technology.

The main challenges we are expected to face are that we might need to code and the definite need to understand the concepts behind how a chatbot learns and communicates. These are the steps we took before configuring each of the chatbots:

1. Educate ourselves and try to understand chatbots
2. Sign up for each chatbot service provider
3. If required, pick the most relevant predefined chatbot configuration
4. Make sure our goal and use case is consistent for each chatbot we configure

My Contribution:

As mentioned above, there were many challenges that we faced while developing the project. My contribution to the entire project was that I was able to extract the App Level code from the base code and PUBLISH the base code as an NPM Package - https://www.npmjs.com/package/manikkhurana-dialog2

3. PROBLEM DEFINITION

3.1 Defining the problem

The main challenges we are expected to face while deciding the framework or algorithms to choose from the various chatbot building service providers are that we might need to code and would need to understand the concepts behind how a chatbot learns and communicates. These are the steps we would need to take before configuring each of the chatbots:

1. Educate ourselves and try to understand chatbots
2. Sign up for each chatbot service provider
3. If required, pick the most relevant predefined chatbot configuration
4. Make sure our goal and use case is consistent for each chatbot we configure

A user has to put in a lot of time and effort just to decide the platform on which to build the chatbot, to get rid of this hassle we built a dashboard that is integrated with Google Dialogflow, IBM Watson and Amazon Lex that would provide the user with the required backend according to need.

3.2 Objectives

The primary objective of the entire project is that the project should be a generic. The User, later on should not be bothered much about the underlying base code and hence the original developer wouldn't be required to make changes again and again. Modular Approch of this entire project is one of the greatest highlights of this product.

To be able to perform the following actions on the Chatbot Service Provider (like Google Dialogflow) platform using the dashboard/platform built (via API Calls)-

1. Create an Agent
2. Create an Intent
3. Create an Entity
4. Create a Context
5. Delete an Intent
6. Delete an Entity
7. Create/Delete EntityGroup
8. Perform CRUD operations on Knowledge bases and Documents
9. Delete Agent

4. ALGORITHM

4.1 Functions

The following are a few functions or terminologies of Google Dialogflow we must be aware with to have a complete understanding of the workflow.

Agents -

A Dialogflow agent is a virtual agent that handles conversations with your end-users. It is a natural language understanding module that understands the nuances of human language. Dialogflow translates end-user text or audio during a conversation to structured data that your apps and services can understand. You design and build a Dialogflow agent to handle the types of conversations required for your system.

A Dialogflow agent is similar to a human call centre agent. You train them both to handle expected conversation scenarios, and your training does not need to be overly explicit. Agents also serve as a top-level container for settings and data:

- Agent settings for language options, machine learning settings, and other settings that control the behaviour of your agent.
- Intents to categorise end-user intentions for each conversation turn. • Entities to identify and extract specific data from end-user expressions.
- Knowledge to parse documents (for example, FAQs) and find automated responses.
- Integrations for applications that run on devices or services that directly handle end-user interactions for you (for example, Google Assistant).
- Fulfillment to connect your service when using integrations.

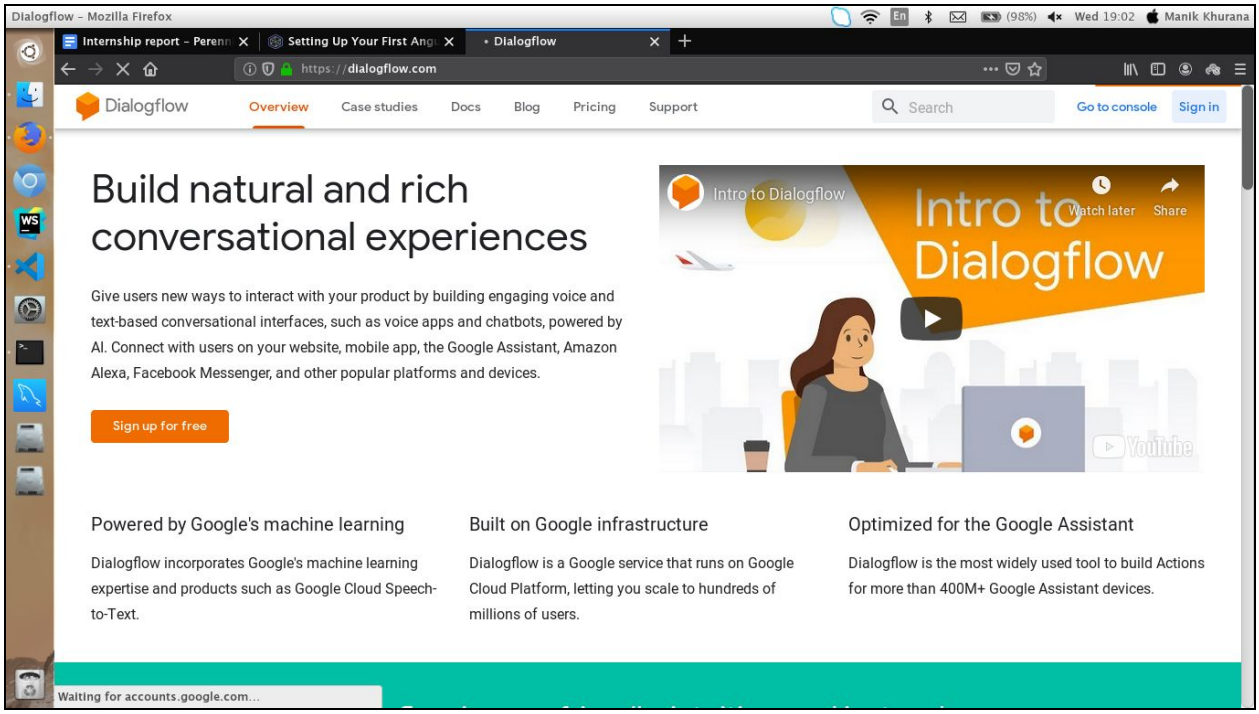

Figure 3: Google Dialogflow

Intents -

An intent categories an end-user's intention for one conversation turn. For each agent, you define many intents, where your combined intents can handle a complete conversation. When an end-user writes or says something, referred to as an end-user expression, Dialogflow matches the end-user expression to the best intent in your agent. Matching an intent is also known as intent classification.

For example, you could create a weather agent that recognises and responds to end-user questions about the weather. You would likely define an intent for questions about the weather forecast. If un end-user says "What's the forecast?", Dialogflow would match that end-user expression to the forecast intent. You can also define your intent to extract useful information from the end-user expression, like a time or location for the desired weather forecast. This extracted data is important for your system to perform a weather query for the end-user.

A basic intent contains the following:

- Training phrases: These are example phrases for what end-users might say. When an end-user expression resembles one of these phrases, Dialogflow matches the intent. You don't have to define every possible example, because Dialogflow's built-in machine learning expands on your list with other, similar phrases.

- Action: You can define an action for each intent. When an intent is matched, Dialogflow provides the action to your system, and you can use the action to trigger certain actions defined in your system.
- Parameters: When an intent is matched at runtime, Dialogflow provides the extracted values from the end-user expression as parameters. Each parameter has a type, called the entity type, which dictates exactly how the data is extracted. Unlike raw end-user input, parameters are structured data that can easily be used to perform some logic or generate responses.
- Responses: You define text, speech, or visual responses to return to the end-user. These may provide the end-user with answers, ask the end-user for more information, or terminate the conversation. The following diagram shows the basic flow for intent matching and responding to the end-user.

A more complex intent may also contain the following:

- Contexts: Dialogflow contexts are similar to natural language context. If a person says to you "they are orange", you need context in order to understand what the person is referring to. Similarly, for Dialogflow to handle an end-user expression like that, it needs to be provided with context in order to correctly match an intent.
- Events: With events, you can invoke an intent based on something that has happened, instead of what an end-user communicates.

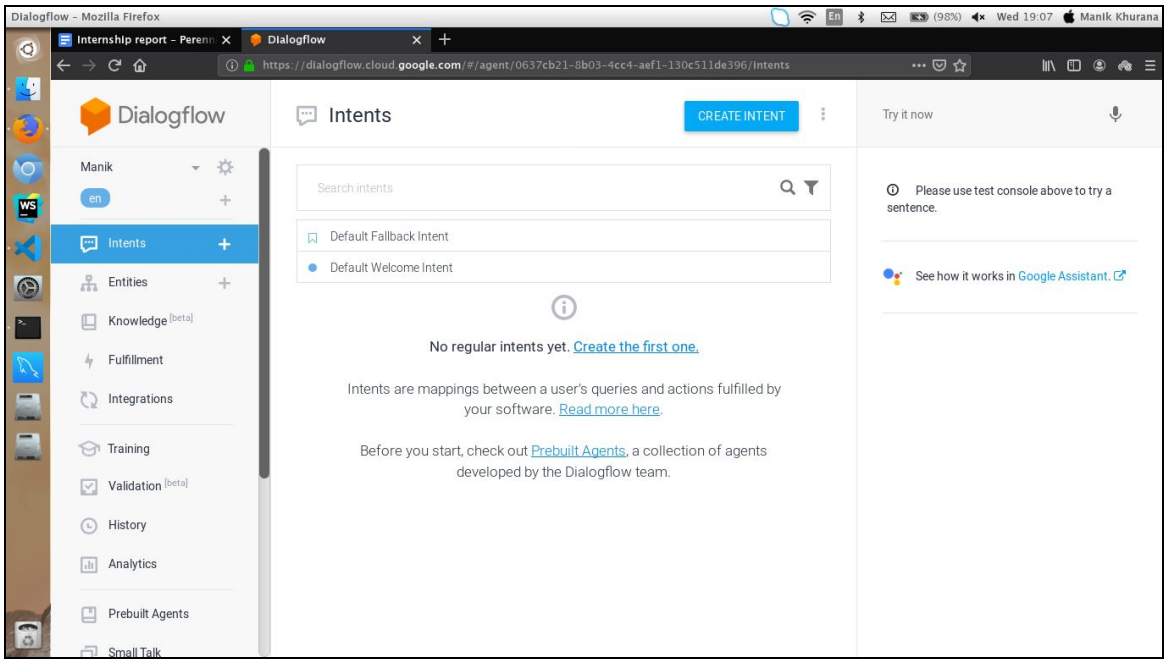

Figure 4: Google Dialogflow: Intents

Entities-

Each intent parameter has a type, called the entity type, which dictates exactly how data from an end-user expression is extracted. Dialogflow provides predefined system entities that can match many common types of data. For example, there are system entities for matching dates, times, colors, email addresses, and so on. You can also create your own developer entities for matching custom data. For example, you could define a vegetable entity that can match the types of vegetables available for purchase with a grocery store agent. Entity terminology The term entity is used in this documentation and in the Dialogflow Console to describe the general concept of entities. When discussing entity details, it's important to understand more specific terms:

- Entity type: Defines the type of information you want to extract from user input. For example, vegetable could be the name of an entity type. Clicking Create Entity from the Dialogflow Console creates an entity type. When using the API, the term entity type refers to the EntityType type.
- Entity entry: For each entity type, there are many entity entries. Each entity entry provides a set of words or phrases that are considered equivalent. For example, if vegetable is an entity type, you could define these three entity entries:
    - carrot,
    - Scallion, green onion
    - bell pepper, sweet pepper
- When editing an entity type from the Dialogflow Console, each row of the display is an entity entry. When using the API, the term entity entry refers to the Entity type (EntityType.Entity or EntityType_Entity for some client library languages).
- Entity reference value and synonyms: Some entity entries have multiple words or phrases that are considered equivalent, like the scallion example above. For these entity entries, you provide one reference value and one or more synonyms.

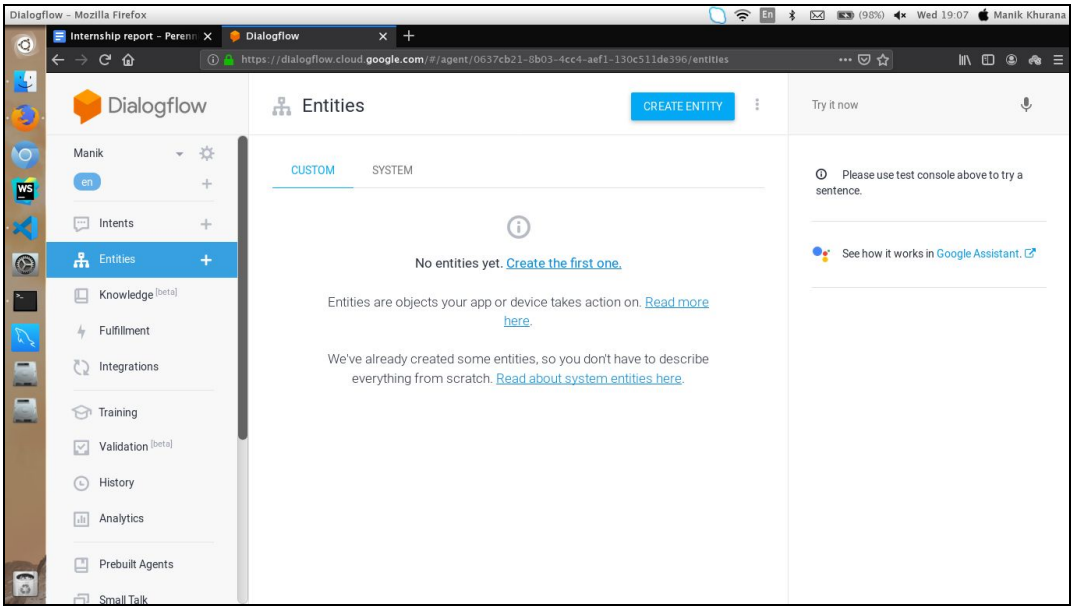

Figure 5: Google Dialogflow: Entities

4.2 Block Diagram / Flow Charts

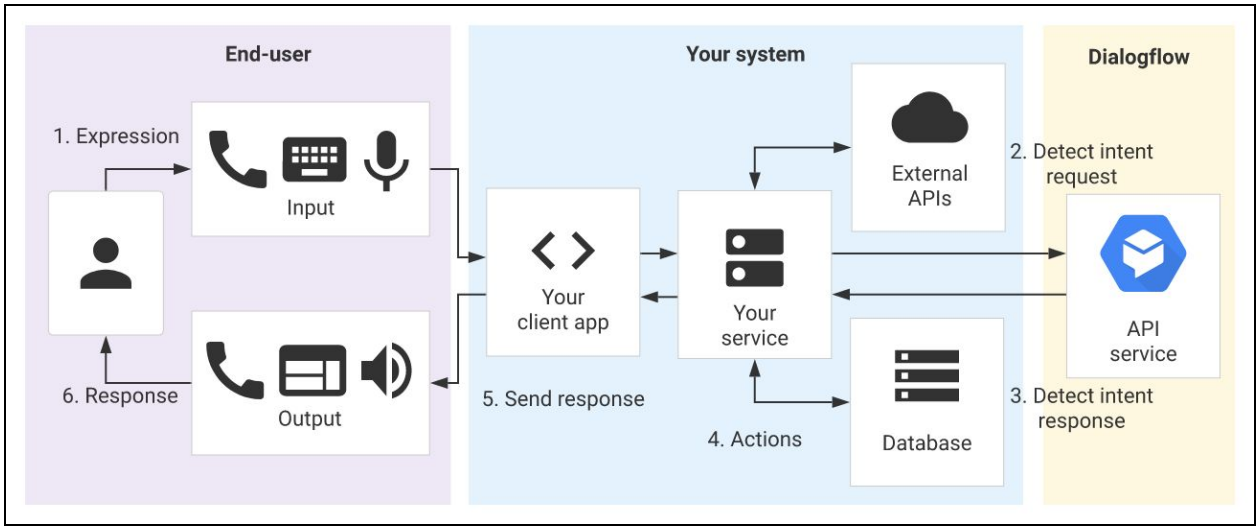

Figure 6: Google Dialogflow: Flow of Control

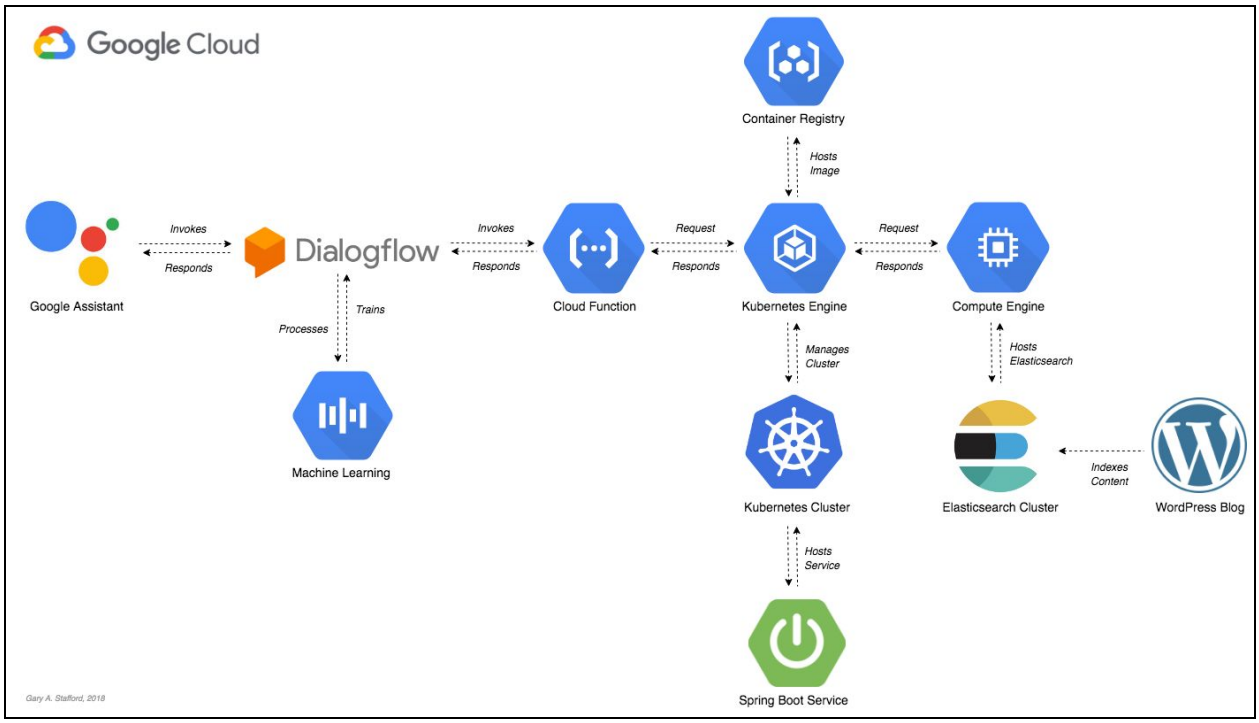

Figure 7: Google Dialogflow: Process Flow

4.3 Algorithm description.

Workflow of the application -

Step I - Create Super Admin.(Google Cloud Platform account owner)

Step 2- Create Agents using middleware.(SuperAdmin)

Step 3- Generation of Admin Credentials.

Step 4- Creation of API keys.

Step 5- Passing of Admin Credentials and API keys to the Admin(User or Agent)

Step 6 - User can perform CRUD operations on intents, entities etc.

4.4 Explanation with example.

Step 1- The product owner(middleware)is the SuperAdmin who will have access to creation ofAgents(clients).

Step 2- Super Admin will create Admin credentials and API keys.

Step 3- After passing of the credentials and API keys the Admin can perform CRUD operations on intents, entities etc.

4.5 NPM Package Published

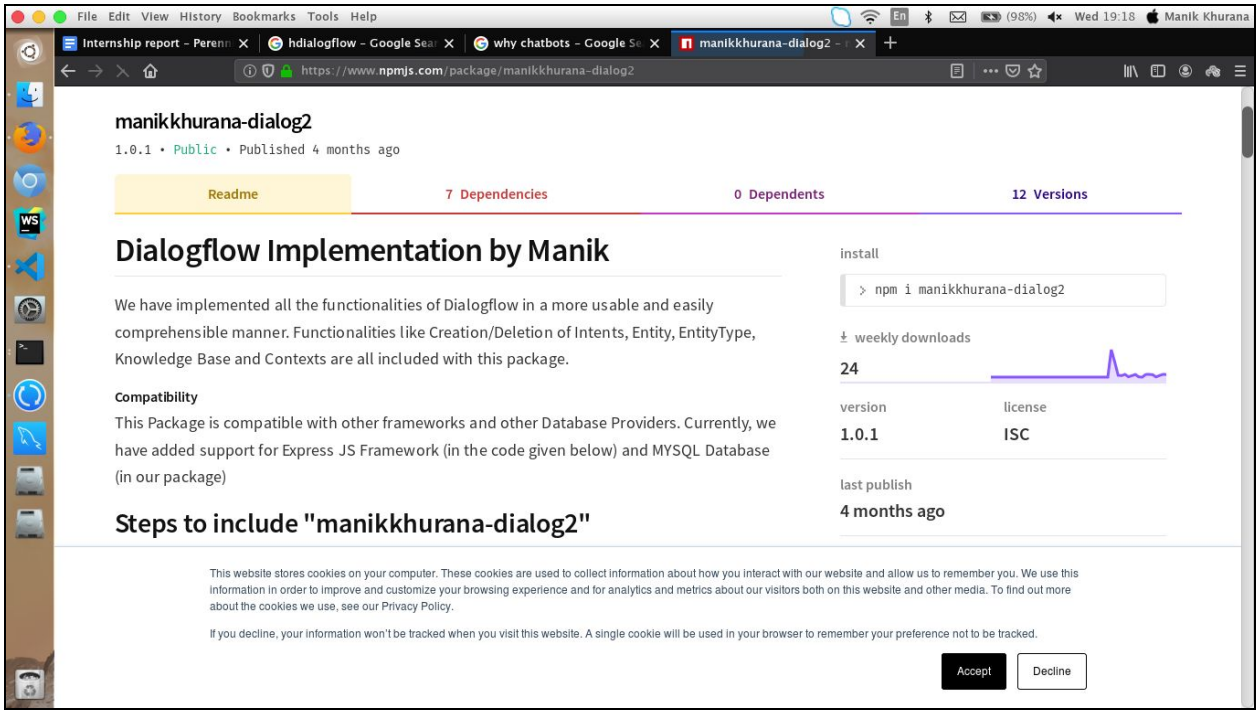

Figure 8: Dialogflow Implementation by Manik

Dialogflow Implementation by Manik

We have implemented all the functionalities of Dialogflow in a more usable and easily comprehensible manner. Functionalities like Creation/Deletion of Intents, Entity, EntityType, Knowledge Base and Contexts are all included with this package.

Compatibility

This Package is compatible with other frameworks and other Database Providers. Currently, we have added support for Express JS Framework (in the code given below) and MYSQL Database (in our package)

Steps to include "manikkhurana-dialog2"

This NPM Package contains libraries pertaining to all the REST APIs required for the aforementioned functions

Follow these steps below in order

- Make sure you have mysql and mysql-workbench installed on the PC
- Create a Database in MySQL via terminal or mysql-workbench as applicable
- Make a new directory myproject and initialize the directory with NPM -

```
$ mkdir myproject

$ cd myproject

$ npm init --yes
```

- Now we have the node_modules folder in place
- Run the following command to create the file index.js in myproject/ and then install Express JS

```
$ touch index.js

$ npm install express
```

- open index.js and paste the following code into it -

```
process.env.sqlDBname = '<Name of the DB you created>';
```

```
process.env.sqluname = '<Your SQL Username>';

process.env.sqlpassword = '<Your SQL Password>';

const code = require ('manikkhurana-dialog2/src/codefile')

const credentials = require ('manikkhurana-dialog2/cred')

const express = require('express');

const router = express.Router();

const app = express();

const bodyParser = require('body-parser')

const cors = require('cors');

router.use(bodyParser.json());

 router.use(function(req,res,next)

{    res.header("Access-Control-Allow-Origin","*");

    res.header("Access-Control-Allow-Headers", "*");

    res.header("Access-Control-Expose-Headers", "token");

    next();

})

var corsOptionsDelegate = function (req, callback) {

    var corsOptions = { origin: true };

    callback(null, corsOptions);
```

```
}

router.use(cors(corsOptionsDelegate))

router.options('*', cors(corsOptionsDelegate))



//Write here: POST/GET Calls



app.use(router)

app.listen(3000, () => {

  console.log("Running on http://localhost:3000")

});
```

- Now you have to open the json file which has all the credentials stored. This is the file you download when you make a project on Google Cloud Platform after agent creation in Google Dialogflow
- To start using the package, we have to make POST calls as mentioned below
- Also, inside the "scripts" tag in package.json, add the "start" tag. This helps you start the server simply by "npm start" in the terminal

```
"scripts": {

   ... ,

   "start": "node index.js"

 },
```

- Type npm start and BAM! your server is ON
- If you want to Use all the functionalities, you can directly copy all these POST Requests and paste it inside the index.js file

```
router.post('/createAgentEntry', code.createAgentEntry)

router.post('/createIntent', credentials.credfunc, code.createIntent); //Write like this one to add functionalities

router.post('/deleteIntent', credentials.credfunc,code.deleteIntent);

router.post('/detectIntent', credentials.credfunc,code.detectIntent);

router.post('/listIntent',  credentials.credfunc,code.listIntent);

router.post('/createEntityType',  credentials.credfunc,code.createEntityType);

router.post('/createEntity', credentials.credfunc, code.createEntity);

router.post('/createKB', credentials.credfunc, code.createKnowledgeBase);

router.post('/deleteKB', credentials.credfunc, code.deleteKnowledgeBase);

router.post('/getKB', credentials.credfunc, code.getKnowledgeBase);
```

Common Errors and how to resolve them

- (node:27919) UnhandledPromiseRejectionWarning: FetchError: request to https://www.googleapis.com/oauth2/v4/token failed, reason: getaddrinfo EAI_AGAIN www.googleapis.com Reason - You are not connected to the Internet
- Unhandled rejection TypeError: Cannot read property 'intentId' of undefined Reason - The name value that you provided in the request doesnot exist in the DB for that particular Agent-Name

Common Understood facts -

- There is no error in the NPM Package. Needn't change anything there.
- You cannot delete what you haven't created
- For creating any Entity you must create an EntityType before
- The /createAgentEntry must be your first call when you create the project. This will store your Credentials in the DB. After this step, you can call APIs anytime by specifying the Agent-Name

5. PERFORMANCE ANALYSIS

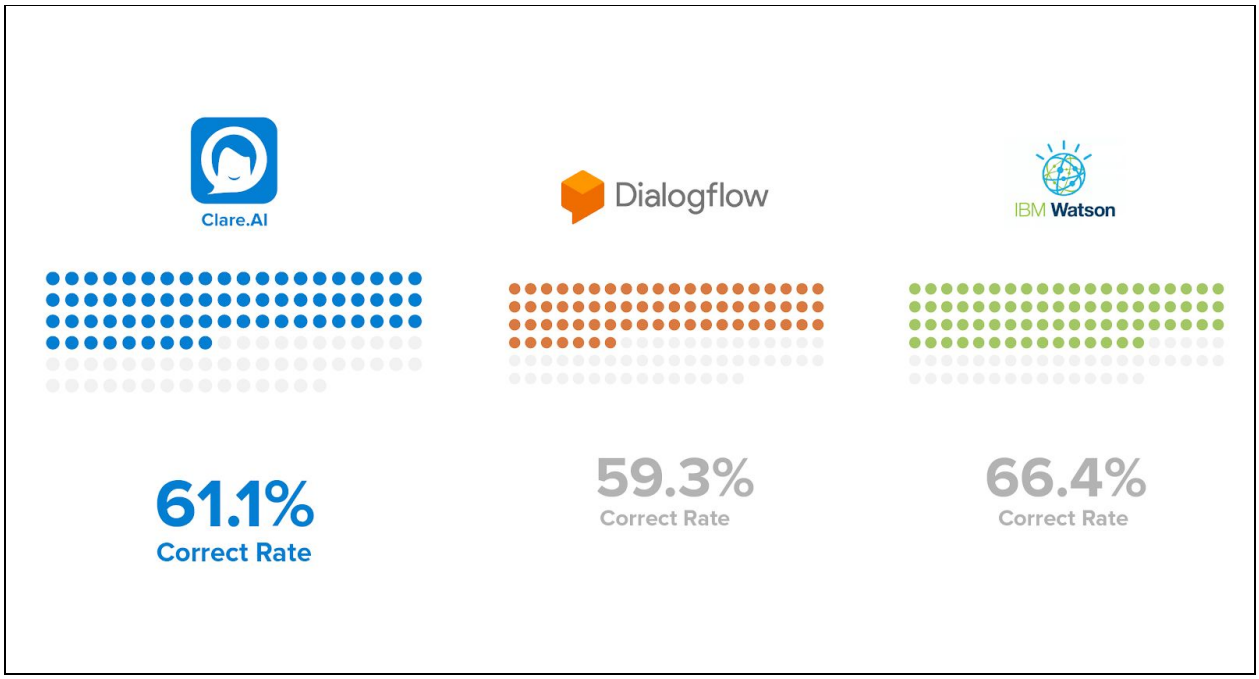

Figure 9: Dialogflow Comparison

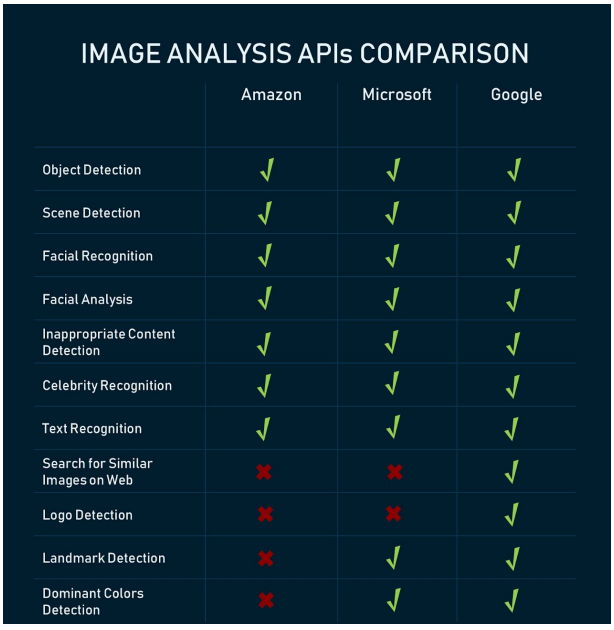

Figure 10: Dialogflow Comparison -2

6. CONCLUSIONS

6.1 Outcomes

As mentioned in the objectives we were able to

1. Publish an NPM Package of the entire base code
2. Create over 12 APIs to call Dialogflow functions
3. Integrate code with an Angular JS based frontend code to facilitate the API calls

6.2 Drawbacks

Our current PoC (Proof of Concept) is highly dependent on Google Dialogflow APIs that have been built to interact with the Dialogflow, a slight change in the APIs, policies, pricing etc. in any platform can impact the application drastically.

6.3 Future

Current PoC is integrated with Google Dialogflow but the other platforms must integrated to fulfil the objectives of the project, completely.

Other features like training, up be loading of test cases etc. from the middleware itself can be done depending on the APIs present.

7. REFERENCES

[1] https://dialogflow.com/

[2] https://dialogflow.com/docs

[3] https://cloud.google.com/

[4] https://www.kommunicate.io/blog/dialogflow-vs-lex-vs-watson-vs-wit-vs-azure-bot/