Serverless Deployment of a Voice-Bot for Visually Impaired

Deepali Bajaj[1], Urmil Bharti[2], Hunar Batra[3], Anita Goel[4], S. C. Gupta[5]
[1, 2, 3]Department of Computer Science, Shaheed Rajguru College of Applied Sciences for Women, University of Delhi, Delhi, India,
deepali.bajaj@rajguru.du.ac.in,urmil.bharti@rajguru.du.ac.in,
i@hunarbatra.com

[4]Department of Computer Science, Dyal Singh College
University of Delhi, Delhi, India, goel.anita@gmail.com

[5]Department of Computer Science, Indian Institute of Technology
Delhi, India, scgupta@cse.iitd.ac.in

Abstract - Serverless Computing is an upcoming paradigm for the development and deployment of cloud applications. Rather than directly provisioning cloud infrastructure, developers write their short, ephemeral functions to be executed on a serverless platform. This simplified programming model relieves its developers from the burden of server management tasks like allocation, scaling and de-allocation of resources. Resources are typically allocated and charged in proportion to the number of times a function call is made. Monetary profits of serverless architectures greatly depend on the function execution behavior and application workloads volumes. Out of few relevant case studies, certain applications are exemplar and suit perfectly to serverless frameworks - like voice bot and chatbot applications. Recently, chatbots are becoming popular in all varieties of business applications as they can handle multiple users' requests at a time and reduce per service cost. In this paper, we have investigated a case study of "*Feed-O-Back* - A Teachers Feedback Voice bot for visually impaired students" and we also observed its implementation in serverless frameworks. Feed-O-Back Bot is a web application powered by conversational AI and enabled with natural language processing to collect the feedback from the students. Since this bot is used intermittently and applicable to few students only with special needs, reserving a dedicated cloud virtual machine instance will not be economically justifiable. We opted for Google Cloud Platform (GCP) and compared pricing of dedicated cloud instances (Infrastructure-as-a-Service model - IaaS) to serverless architectures. We evaluated that Google Cloud Function (FaaS model) is more cost-efficient and proven to be economical than Google Compute Engine (IaaS model). We have described the details of voice bot implementation using Dialogflow conversation service deployed on Cloud Function for Firebase.

Keyword: Serverless, Function as a Service, Voice bot, Google Cloud Platform, Google Compute Engine, Cloud Function for Firebase, Heroku, Cloud Firestore

1. Introduction

Serverless computing is a new trend in software development. This paradigm provides an opportunity to its developers with an event driven programming model for developing cloud applications and abstracts away provisioning of infrastructure and management of compute resources [1]. It charges its users only for resources that are consumed for executing their functions rather than dedicated resource allocation. In comparison to Infrastructure-as-a-Service (IaaS) model where virtualized compute resources are typically billed on hourly quanta (or larger), serverless platforms are charged at a very granular level nearly 100-millisecond level. In a serverless model, idle or underutilized resources are never billed to the users. Thus serverless offers a "zero use = zero cost" model [2]. These evident benefits led various cloud infrastructures providers to bring up their serverless computing capabilities. Presently AWS Lambda [3], Microsoft Azure Functions [4], Google Cloud Functions (GCF) [5], and IBM OpenWhisk [6] are prominent and mature serverless platforms.

Applications implemented on serverless models are capable of scaling almost instantly i.e. from tens to thousands of concurrent invocations of a serverless function are supported. This enables serverless platforms to scale and meet truly unpredictable loads of any spikes and surge [7]. This very characteristic makes serverless functions suitable for developing chatbots. There is practically no limit to the number of requests a serverless function can handle, making serverless a strong contender for chatbots [8].

Fundamentally, Serverless Functions are priced on three parameters i) function execution duration - how long function runs (in milliseconds) ii) Number of executions (requests) i.e. how many times a function is invoked iii) Memory Size - how much memory reserved for the function. Thus some decent workloads that best suits serverless framework are as follows [9],[10]:

i) Bursty workloads where the functions can also scale to zero, so there is no cost to the consumer when the system is idle.
ii) Any important but infrequent business function/requirement
iii) "Narrow" applications / simple and short-term workload which consist of little more than business logic over a database.

Using dedicated infrastructure for above mentioned workloads would be financially unreasonable, so serverless models are ideal for their implementation [11].

Battle for public cloud service provider dominance is a tripartite contest among Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). Though AWS is the oldest and most experienced player in the cloud market, GCP is also performing well and is relentlessly working to become top cloud based provider [12]. GCP is looking promising in terms of growth rate and pricing models [12]. An earnings report indicates that GCP observed a steep growth from 2018 to 2019. According to Canalys report - February 2020, that annual growth of Google cloud is 67.6% in comparison to AWS 33.2% and Microsoft Azure 62.3% for Q4 2019 [13]. In fact, in terms of monetary comparisons, GCP is providing more customer friendly pricing and discount models to its users [14] as compared to other serverless providers. Figure 1 shows a sample price comparison (per month) of both GCP and AWS platforms for

various cloud services. Comparison shows GCP is more money-saving proposition and for this reasons we performed our experimental work and comparison on two service models of GCP i.e. Google Cloud Functions [5] and Google Compute Engine [15].

To evaluate our concept, we intend to develop a web application *Feed-O-Back* – A Teachers' Feedback Voice bot for visually impaired students. Students use this voice bot application to record their feedback at the end of every semester for all teachers. Feedback is collected on fourteen different questions that are based on parameters important for improving the overall education system. Students can mark their response on a scale of 1 to 5. From the implementation perspective, it will be a voice bot application that is being developed using conversation technology via audio messages. We used Google Dialogflow [16] conversational service to develop our voice bot application and deployed it on Google Cloud Function for Firebase and integrated with Google Cloud Firestore database. The web application invoking this bot facility is deployed on Heroku [17] which is a cloud application platform used to build, deliver, monitor and scale cloud applications.

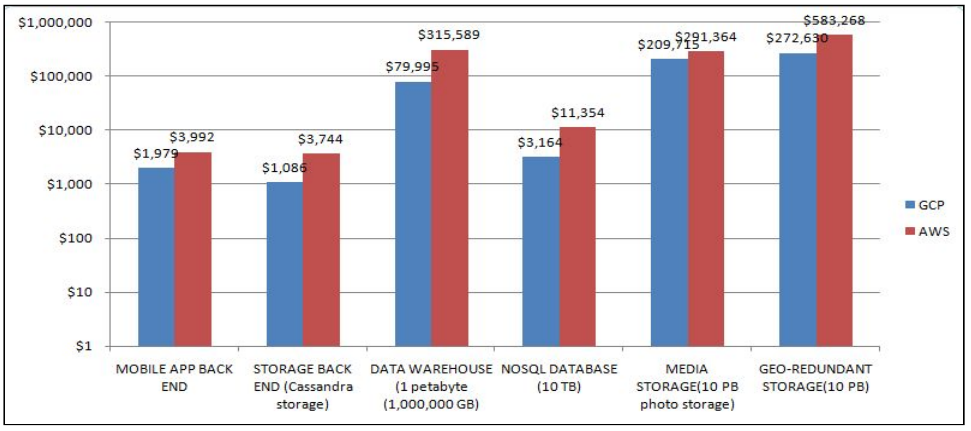

Fig 1. A sample price comparison of both GCP and AWS platforms

Major contributions of the paper are:

C1: A comparison of pricing strategy for both service models i) FaaS - Serverless architecture (GCF) and IaaS – dedicated virtual machine instances architecture (GCE)

C2: Implementation of *Feed-O-Back* – A Teachers' Feedback Voice bot using Google Dialogflow and deployed on GCF. Web app invoking this voice bot utility is deployed on Heroku Cloud Platform [https://feedbackbottest.heroku.com].

C3: Benchmarked the performance of Teachers' Feedback Voice bot using Apache Bench tool.

Rest of the paper is organized as: Section 2 is related work where we have discussed our literature review and understood the deployment aspects of existing bot applications. Section 3 compares pricing strategy for both service FaaS and IaaS of Google Cloud Platform (C1). Blueprint describing design and implementation details of "*Feed-O-Back* - Serverless Voice Bot for Visually

Impaired" is discussed in section 4 (C2). Section 5 depicts results of performance benchmarking of Teachers' Feedback Voice bot (C3).

2. Related Work

Chatbot/Voice bot has been around for almost ten years. But recently it is gaining widespread popularity due to technological compatibility with serverless frameworks.

Luo et al. [18] performed an experimental study to assess the goodness of chatbots. They experimented on 6,200 customers who receive calls from chat bots or humans. Their results show that chatbots are four times more effective in comparison to inexperienced workers. They also claimed that chatbots can be as effective as proficient workers. Serban et al.[19] presented a MILABOT - ensemble-based dialogue system. This bot was skilled to converse with humans on popular topics via both speech and text and leverages deep learning and reinforcement learning techniques. Xu et al. [20] developed a chatbot for evaluating customer service on twitter. They evaluated that their bot is as good as human agents in expressing empathy to assist users needing emotional support. Results also showed their bot performed better than the information retrieval system. Yan et al. [21] present the architecture and prototype model of a chatbot based on a serverless platform implanted on IBM Watson Developer Cloud. Ranoliya et al. [22] - gave a design of a chatbot for university related FAQs using Artificial Intelligence Markup Language. Their study does not discuss deployment issues and database handling for their chatbot. Kumat et al. [23] developed an Android based educational chatbot to handle queries of the visually impaired people via text and speech both. But their paper is devoid of deployment details so does not bring clarity on platform related issues. Another personal assistant Android based app focusing on image recognition, currency recognition, and e-book capability is described by [24]. Metatla et al. [25] developed a Voice user interface (VUI) for visually-impaired to be used in schools for inclusive education.

Since the conducted literature review reveals a mixed representation of chatbot and voice bot applications, most have not elucidated deployment details and cloud based database support for their bots. Some of them have opted for dedicated cloud instances making them overly pricey for sporadic applications. Few researchers have designed their chatbot on serverless platforms like AWS and IBM but do not clearly represent performance, scalability, availability and latency details of their applications. Thus we argue that there is a need to understand the economics of voice bot applications on serverless platforms. For our study we designed and developed a voice bot application to deploy on pay-per-use model of serverless frameworks to understand its cost benefits. Thus, serverless platforms

can offer interesting opportunities for future chatbot /voice bot applications which is largely unexplored yet.

3. Background

According to a research study done by Forrester, "Globally, either 57% of companies are using chatbots already or planning to do so in the coming years [8]. Experts think that the use of chatbots is going to explode in near future. Serverless technologies make chatbots easier to build as these platforms provide developers with many built-in building blocks for creating them inexpensively [26].

Conceptually, a bot can execute automated tasks like booking a hotel, ordering cakes and bouquet, ordering food, and so on. They are powered by advanced artificial intelligence capabilities like Natural Language Understanding (NLU) and Automatic Speech Recognition (ASR). Chatbots and Serverless platforms are like a match made in heaven. Out of the diverse range of successful use cases for serverless frameworks, chatbots/ voice bots are a good fit for this technology. For these reasons, most of the enterprise cloud service providers are coming up with their serverless conversation service development frameworks as listed below:

- Dialogflow (Previously known as API.AI) is Google service which is a natural language understanding (NLU) platform used to build "build once deploy everywhere" conversational interfaces for mobile apps, web applications, IOT devices, bots, interactive voice response systems, and so on [16].
- AWS Amazon Lex is a service for designing conversational interfaces for applications including both voice and text. Amazon Lex enables developers to build high-end, natural language chatbots/voice bots into a new or an existing application [27].
- Microsoft LUIS is a cloud-based Language Understanding (LUIS) service that is used to develop social media apps, chat bots, and speech-enabled desktop applications. It coalesce machine learning techniques to conversational applications that communicate with a user in natural language to complete a task [28].

3.1 Economics and Architectural impact of serverless Applications

To analyze and assess economics, we compared pricing of serverless infrastructure with serverful IaaS using automated online calculators. We evaluated Google Compute Engine (GCE) and Google Cloud Functions (GCF) of Google Cloud Platform (GCP). GCE is the compute infrastructure component which permits users to launch virtual machines (VMs) on demand. Google Cloud Functions (GCF) which is a serverless platform where Google Cloud handles the operational infrastructure on behalf of its users. In order to model Teachers' voice bot application for both GCF and GCE architectures, we need to set a few technical assumptions on some parameters. For serverless architectures (GFC), significant parameters are a) total number of requests handled by bot b) request duration in milliseconds c) memory requirement for each service request d) requests distribution over time. Similarly for reserving a dedicated cloud machine instance (GCE) significant parameters are a) machine type and family b) region c)

special Operating and software requirements for machine instance d) average number of hours/days machine instance will be used.

- For GCF implementation, we used Serverless Cost Calculator [29] that is used to calculate cost for various serverless platforms. We argue that this kind of voice bot application will be invoked sparsely and only for the students with special needs. With this notion, we assume i) Number of executions will lie between 100-1000 ii) Estimated execution time will take 100 -1000 ms iii) memory size = 1024MB. Results obtained from Serverless Cost Calculator are shown in figure 2.
- For GCE implementation we assume that this would incur 24/7-equivalent usage charges for the number of instances where voice bot application is live and deployed on GCP infrastructure. GCP offers different options for memory and processing power capabilities. To get an educated guess on price, we use Google Cloud Pricing Calculator [30] that gives a quick estimation of cost on GCP based on usage statistics. Here we assume i) Number of instances = 1 ii) SQL Server Web as software requirement iii) Machine Class – regular iv) Machine family – General Purpose v) Series – E2 vi) Machine Type – e2-standard-2 vii) Region – Mumbai (asia-south a) viii) Average hours per day each server is running – 8 hours per day ix) Average days per week each server is running – 5. Results obtained from Google Cloud Pricing Calculator for GCE are shown in figure 3. *e2* instance is a shared core machine type having 2 v CPUs and 8GB RAM mainly used for short periods of bursting [31]. It is useful for day-to-day computing at a lesser cost and mainly used for web serving, app serving, back-office applications, small to medium databases and microservices types of workloads.

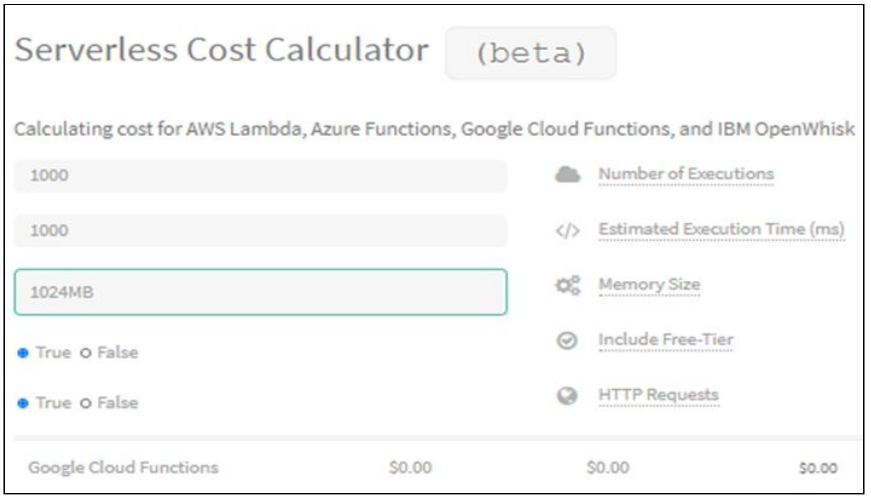

Fig 2 : Serverless Cost Calculator for GCF

We understand that for GCE, developers need to set up a dedicated server/ a cluster of servers, API gateways, load balancers, manage identity and access control mechanisms and many more. These actions would be costly, painful and will take a great deal of time in comparison to serverless deployment options.

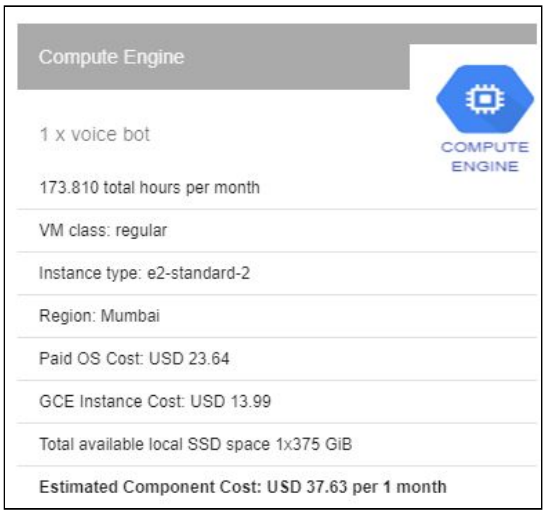

Fig 3 : Google Cloud Price Calculator for GCE

Comparison between serverless and dedicated cloud instance architecture indicates that Feed-O-Back application implementation would be more cost effective on GCF rather than GCE.

3.2 General Bot Architecture

Bots can be classified as: i) Scripted bots and ii) Conversational bots. Scripted bots are pre-defined with a conversational flow. When a user sends a request, the bot responds based on a pre-defined knowledge base. The bot cannot go out of the defined scope. A scripted bot is usually helpful when we want to automate a task like filling a grievance, submitting a feedback etc. Conversational bots use the power of NLP, NLU and NLG and help its users to get the answers of their queries. They enable users to ask any question. These types of bot are trained to respond to user's queries with the most reliable response.

For any bot, text and voice are the main types of input data that can be sent by a user. Text was the most widely used input for chatbots. Nowadays, voice bots are getting popular. In this, users provide a voice input via a voice assistant. This capability is helpful to enhance the accessibility of voice bots for people with visual disabilities. For a voice bots following software are required - i) Speech recognition ii) Speech-to-text and iii) Text-to-speech.

Once the human chat is collected, it should be preprocessed using NLP libraries to handle errors like Spell check, Upper/lower case, Split into sentences, Removing noise & stop words, Lemma-tise words to find canonical forms of a word like pay, paid and paying etc. This step will yield a list of clean words contained in each sentence. Next step is Intent Classification which is more critical step for voice bot/chatbot applications as it is subjected to two errors – a) False positive - user does not state an intent however the chatbot recognizes an intent b) False negatives - user states an intent however the chatbot is unable to find it.

Intent classification can be either pattern matching or NLU. Pattern matching makes use of pattern match to group text and create a suitable response from the user. For each query, a pattern is available in the knowledge base to provide an appropriate response. Artificial Intelligence Markup Language is a standard model to specify these patterns. Natural Language Understanding is the most popular method of intent classification. Once the intent is matched, the Natural Language Generator produces a response and returns it back to the user. Bots are also connected to the Database/ knowledge base or external APIs to generate a more appropriate dynamic response.

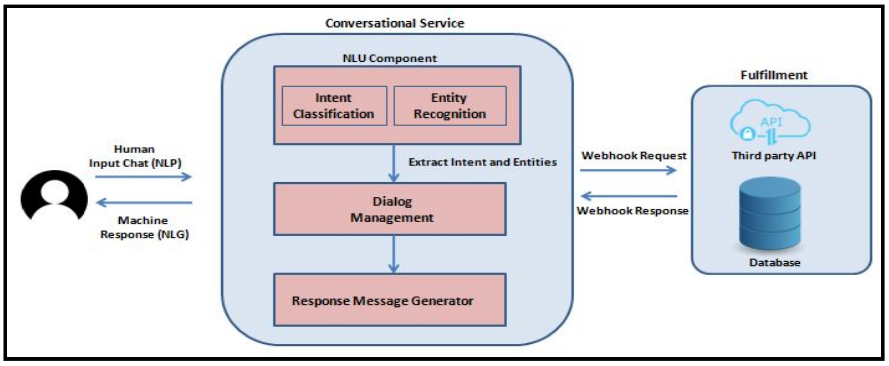

Fig 4: Generic Bot Architecture

4. Blueprints for a Serverless Voice Bot for Visually Impaired

In our case study, to submit feedback, students can simply open the Feed-O-Back Bot web application and interact with the voice-enabled personal assistant to provide their feedback for the teachers of the current semester. The conversational agent of Feed-O-Back bot is voice-based and it interacts with the users in natural language, thus, making the feedback collection process more interactive, simple and easy. The flow of the conversation consists of questions asking the user for their name, department, year and then the teachers name and the subject taught. After this basic information about the user is collected, next fourteen questions are asked which are to be answered on a scale of 1-5 for the respective teacher.

4.1 Overview of Technology Stack for Voice Bot

A Technology Stack is the bundle of technologies that are used to build a voice bot application. It groups Front-end technologies, Backend Technologies, API's and Libraries, Database and Webhook Fulfillment. Table 1 shows the complete technology Stack used for Feed-O-Back Voice Bot.

Elements of a VoiceBot	Software Products and Frameworks used
Front-end Technologies	• HTML (Hypertext Markup Language) • CSS (Cascading Styles Sheet) • Javascript
Backend Technologies	• Node.js

API's and Libraries	• Socket.IO - To enable real-time bidirectional communication between the server and the browser • Web Speech API – For Speech Recognition & Speech Synthesis Interface • Dialogflow (API.AI) v1 NLP API
Database	• Firebase Cloud Firestore - Google cloud-hosted database store the students' feedback
Webhook Fulfillment	• Google Cloud Functions for Firebase - A serverless framework that enable its users to run backend functions in response to events • Node.js 8

Table 1: Technology Stack used for Feed-O-Back Voice Bot

Before getting to the explanation of architecture of Feed-O-Back, below is the brief description of various important elements of a Feed-O-Back voice-bot:

Web Speech API - In order to understand the cues spoken by a user, we have used the Web Speech API's *SpeechRecognition* interface, which provides the ability to recognize context from an audio input. Web Speech API enables bot developers to provide speech-input and speech-output features in the web browser. The Web Speech API has two parts: *SpeechSynthesis* (Text-to-Speech), and SpeechRecognition (Asynchronous Speech Recognition). In Feed-O-Back bot, we have used the *SpeechSynthesis* interface to detect the user's voice input and convert the speech to text and we have used the *SpeechSynthesis* interface to return the response back to the client with a synthesised speech i.e. text to speech as shown in figure 5.

This process of conversion of user speech/utterances to a text string is also known as Speech-to-Text (STT) conversion. The interface's constructor has been used to create a new SpeechRecognition object, which has a number of event handlers available for detecting when speech is input through the device's microphone. The *SpeechRecognition* interface of the Web Speech API is the controller interface for the recognition service; this also handles the *SpeechRecognitionEvent* sent from the recognition service. Once the speech recognition has started, the result event of *SpeechRecognition* is used to retrieve the user's speech input in the form of text. This returns a *SpeechRecognitionResultList* object which consists of the result and the text has been retrieved in the array. It also returns a confidence score for the user's speech transcription detection accuracy.

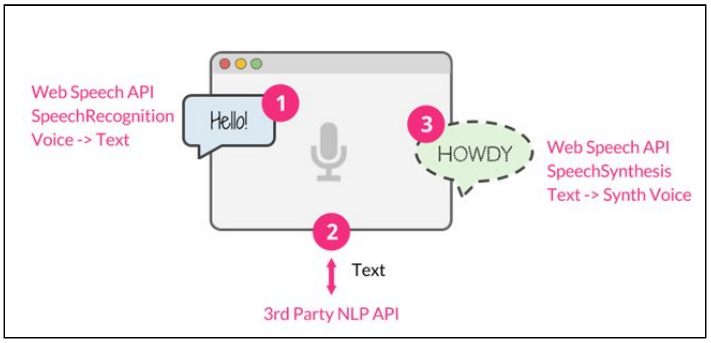

Fig 5: Web Speech API (for speech recognition and speech synthesis)

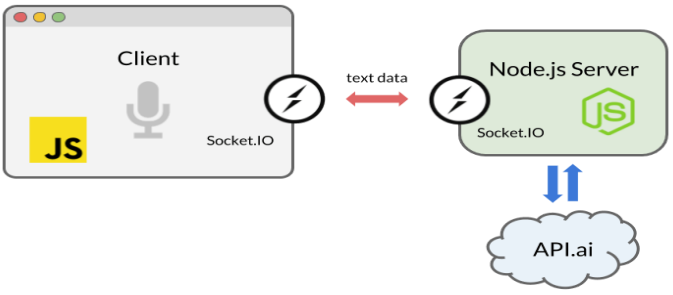

Fig 6: Bi-directional communication by Socket.IO between client and server

SpeechSynthesis interface of the Web Speech API is a text-to-speech component that allows voice bots to read out their text content, normally via the device's default speech synthesizer. The *SpeechSynthesis* interface is the controller interface for the speech service; this can be used to retrieve information as speech. This returned response from the server is sent through Socket.IO back to the client, where the text response is returned to the user by first converting the text to speech and synthesizing the speech. To perform the speech synthesis of the returned response, the text string response is passed as an argument and the *SpeechSynthesis* interface enables the browser to speak the text.

```
recognition.addEventListener('result', (e) =>
  console.log('Result has been detected.');

  let last = e.results.length - 1;
  let text = e.results[last][0].transcript;

  outputYou.textContent = text;
  console.log('Confidence: ' + e.results[0][0]

  socket.emit('chat message', text);
});
```

```
function synthVoice(text, callback) {
  const synth = window.speechSynthesis;
  const utterance = new SpeechSynthesisU
  utterance.text = text;
  synth.speak(utterance);
  utterance.onend = function(event) {
    recognition.start();
    callback('ok');
  }
}
```

Socket.IO which was created in 2010 is a JavaScript library for real-time web applications. It enables real-time bidirectional communication between web clients and servers as shown in figure 6. It has been used to pass the result from the browser to the backend Node.js code, and then pass the response back to the browser. Web Socket via Socket.IO has been used because sockets are the best solution for bidirectional communication over HTTP or AJAX, especially when pushing an event from the server to the browser. With a continuous socket connection, there won't be any need to reload the browser or keep sending an AJAX request at a frequent interval. Figure 6 depicts the bi-directional communication enabled by Socket.IO between the client and server. To listen to the result event from the *SpeechRecognition* interface we place the following line of code in the result event listener.

Once a connection is established between the client browser and the server using Socket.IO, we use Dialogflow API.AI to process the users message and retrieve a reply and when Dialogflow API.AI returns the result, Socket.IO's *socket*.emit() is used to send the reply back to the client browser.

```
io.on('connection', function(socket) {
  socket.on('chat message', (text) => {
    console.log('Message: ' + text);

    // Get a reply from Dialogflow API.ai

    let apiaiReq = apiai.textRequest(text, {
      sessionId: APIAI_SESSION_ID
    });

    apiaiReq.on('response', (response) => {
      let aiText = response.result.fulfillment.speech;
      console.log('Bot reply: ' + aiText);
      socket.emit('bot reply', aiText);
    });

    apiaiReq.on('error', (error) => {
      console.log(error);
    });

    apiaiReq.end();
```

Dialogflow (API.AI) – Dialogflow is a NLU based platform that makes it easy to design and integrate a conversational user interface [32]. It is used for natural language processing in Feed-O-Back bot. The users speech input detected by the Web Speech API's *SpeechRecognition* interface is sent to a Dialogflow in the form of a text string for further processing of the user input. To use the Dialogflow API.AI in our web application, Dialogflow's Node.js SDK has been used. Through the Dialogflow NLU capabilities, it maps the user query to the one of the matched intent and extracts structured data. For intent mapping, Dialogflow agent utilizes a hybrid model involving two machine learning algorithms - Rule-based grammar matching and ML matching.

To recognize user expressions, Dialogflow applies both of these algorithms and picks the most suited result. The annotated training phrases corpus is used to train Dialogflow agent. To begin with, the hybrid model tries to find intent match using rule-based grammar. If there is no match is, it shifts to ML matching. Thus this mode achieves best and optimized match for most user expressions. To find a matching intent, Dialogflow gives scores to the probable matches by attaching an intent detection confidence value (confidence score) ranging between 0.0 and 1.0. The 0.0 score indicates a completely match uncertainty and 1.0 indicates that the match is completely certain. To filter out false positive, we assign the ML classification threshold value to 0.3. If the confidence score is lesser than the classification threshold value, a fallback intent is triggered. If no fallback intent is defined, then none of the intents is triggered.

Web Hook Fulfillment - To take actions based on end-user expressions and to send dynamic responses back to the end-user, bot uses fulfillment and is integrated with external APIs or databases via a webhook request. This *Fulfillment* can be enabled for each intent. If a matched intent needs some action dynamic response by the system, we should enable fulfillment for that intent. Without fulfillment enablement, Dialogflow generates the static response that we define for the intent. For our bot, we use Cloud Functions for Firebase which is a serverless framework that allows running backend code automatically in response to events triggered by Firebase. The Node.js fulfillment code for Dialogflow processing is stored in Google's cloud.

Database – For our bot, we used Cloud Firestore database [33]. It is a Google cloud-hosted, NoSQL database and is used to store the students' feedback. On Cloud Firestore, a collection was created with 19 fields to record the response of users for their details, feedback for 14 questions and a calculated average score.
To write data to the Cloud Firestore, a document is to be written in the created collection. For adding data, the add ( ) method has been used over the doc ( ) method to avoid overwriting the data in the database. Through add ( ) method, Cloud Firestore auto-generates an ID for the document. Figure 7(a) shows database design of voice bot in Cloud Firestore and figure 7(b) shows the code snippet storing feedback data as a document in the collection on Cloud Firestore.

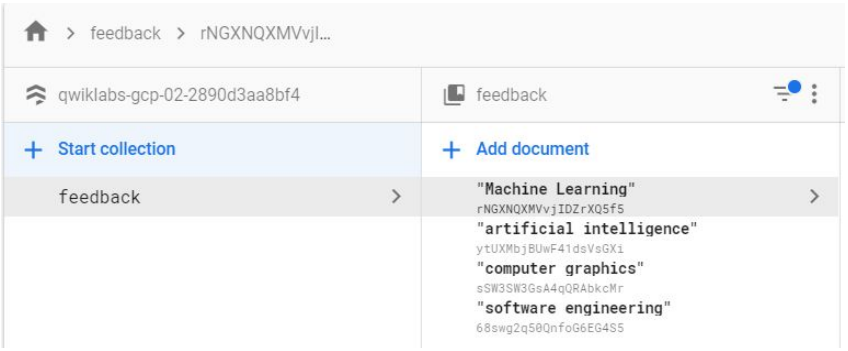

Fig 7(a): Database of voice bot in Cloud Firestore

The web application utilizes HTML, CSS and JavaScript for the front-end framework and Node.js for the backend framework i.e the server. To run the server locally, Express, which is a Node.js open-source web application server framework, was used. It is used to build web applications and APIs. The user interface of the web application has been designed to look intuitive and it consists of a microphone button which is pressed to input speech by the user. The conversation between the user and the bot are also displayed on the screen. To deploy the web application, Heroku has been used.

4.2 Overall Design Architecture of Feed-O-Back Voice bot

The conversational agent of Feed-O-Back Bot is adorned with AI-capabilities and based on a serverless architectural design. It is powered with NLP and NLU to identify the user's query expression and return an appropriate response. For Feed-O-Back application, we used Cloud Functions for Firebase as our backend infrastructure. When a user makes an audio query, it is converted into text using *SpeechRecognition* interface (Speech-to-Text). In the second step, this extracted text is sent to the Dialogflow through *Socket*.IO. NLU is performed on this text to deduce the semantic meaning of the user expressions. On the basis of user utterance semantics, entities are detected, which are mapped to pre-defined intents on Dialogflow. The intent mapping is done by the Dialogflow agent which is trained extensively on annotated corpus of training phrases to enable it to find inferences.

```
db.collection('feedback').add({
    studentName : name
    studentdepartment : dept
    studentYear : year
    teacherName : teacher
    subject : sub
    ques1 : q1
    ques2 : q2
    ques3 : q3
    ques4 : q4
    ques5 : q5
    ques6 : q6
    ques7 : q7
    ques8 : q8
    ques9 : q9
    ques10 : q10
    ques11 : q11
    ques12 : q12
    ques13 : q13
    ques14 : q14
    teacherAvg : avg
});
```

Fig 7(b): Feedback data as collection on Firestore

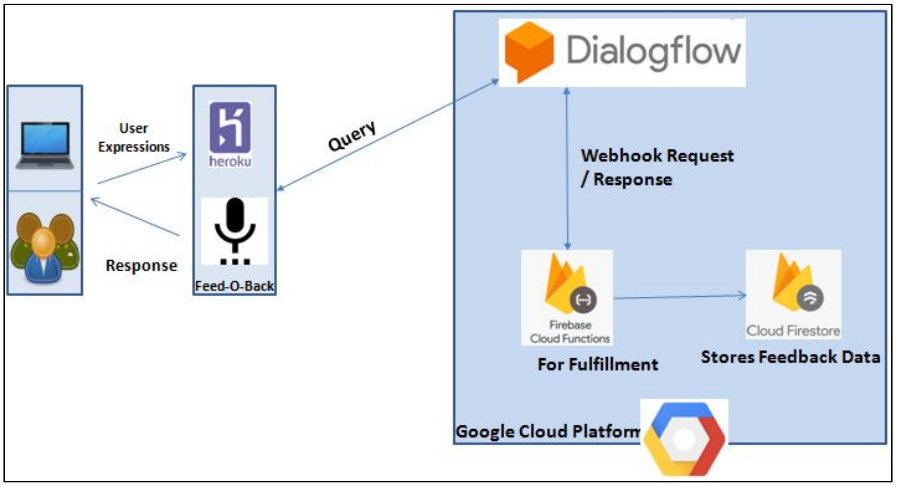

Fig 8: Complete architecture of Feed-O-Back Voice Bot

Our bot application's conversational design consists of 48 intents and each one of them is extensively trained with multiple user utterances. Having understood the intent of the users' query, Dialogflow picks the specific actions to fulfill the intent. An HTTP POST request is sent by Dialogflow for fulfillment to the Webhook to respond users query. For the webhook, Cloud Functions for Firebase has been used. A webhook written in Node.js 8 has been created and deployed on the Google Cloud Functions for Firebase which processes the request, saves the

data to the Cloud Firestore and then sends a response for the request to the server from Dialogflow and then to the client via Socket.IO Further, the final text response is again converted to speech (Text-to-Speech) and synthesized through the *SpeechSynthesis* interface and then returned to the clients browser. Complete architecture of Feed-O-Back Voice Bot is depicted in figure 8.

5. Experimental Results

Testing voice bot means testing a Dialogflow fulfillment to determine how webhook performs under unusual load testing scenarios [34]. We performed an integration test that includes webhook, Dialogflow intent and entity extraction as well. We used the open source Apache Bench (ab) tool [35] for our experiments which is a command line tool for load testing and benchmarking tool for Hypertext Transfer Protocol (HTTP) server. With this tool, we can rapidly assess how many requests our web server can serve per second.

We performed load testing on webhook fulfillment to figure out performance issues that may cause interruption of fulfillment service. Typical load testing scenarios are - a) spike test: unexpected increase in load and b) soak test – continuous sustained loads. For Spike testing, we send a steady number of requests to the webhook for some time and abruptly amplify the load. i.e. sends a load of 10 requests per second with a few spikes of 60 requests per second. For Soak testing, we observe the response by sending a steady number of requests to the webhook for several minutes, i.e. sends a steady load of 10 to 20 requests per second for several minutes to analyze if response times increases.

Following table gives an idea about testing webhook (https://us-central1-qwiklabs-gcp-02-2890d3aa8bf4.cloudfunctions.net/ dialogflowFirebaseFulfillment) under various testing scenarios for Feed-O-Back:

Load Testing of voice bot deployed on Heroku	$ ab -n 100 -c 10 https://feedbackbottest.herokuapp.com
Load Testing of webhook	$ ab -n 100 -c 10 https://us-central1-qwiklabs-gcp-02-2890d3aa8bf4.cloudfunctions.net/dialogflowFirebaseFulfillment
Spike Testing of webhook	$ ab -n 60 -c 60 -p ActionRequest.json -T 'application/json' https://us-central1-qwiklabs-gcp-02-2890d3aa8bf4.cloudfunctions.net/dialogflowFirebaseFulfillment
Soak Testing of webhook	$ ab -t 120 -n 1200 -p ActionRequest.json -T 'application/json' https://us-central1-qwiklabs-gcp-02-2890d3aa8bf4.cloudfunctions.net/dialogflowFirebaseFulfillment

Table 2: Testing Scenarios for voice bot and webhook fulfillment

We plot the output of Apache Bench to observe the time the server (webhook) takes as the number of requests increases. We have plotted the *time* (total time -in ms) with respect to the *number of requests*. We observed that for the initial 80 requests, the total time was nearly 400 ms. For the next 20 requests, it increased to 1600 ms and so on. After running load tests on ab tool, we observed results for webhook response times and analyzed the median response time for load, spike and soak test were 275 ms, 3052 ms and 2225 ms respectively. This suggests our voice bot performs reasonably well under unexpected load of sudden spike as well.

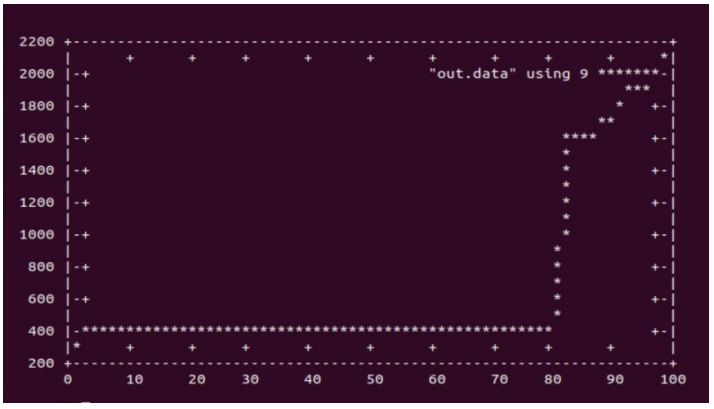

Fig 9: GNU Plot of Load Test on Apache Bench Tool

Conclusion:

Serverless platforms have changed the economics of cloud applications hosting by allowing users to pay only for actual resource utilization and not for reserved capacity. This model is most appropriate for applications that do not require instances running 24/7. Serverless functions automatically scale to support the rate of incoming requests without needing to configure anything. This opportunity makes serverless perfect for voice bot applications. With a rise of serverless architectural frameworks, chatbots / voice bot applications are getting easier to develop. Thus, serverless platforms can offer interesting opportunities for future chatbot / voice bot applications.

Bearing this in mind, we did cost benchmarking to compare the economic impact Teachers' Feedback Voice bot application on both serverless architecture and dedicated virtual machine instances. Since application will be only used for visually impaired students, it would not be economically justifiable to have a dedicated service instance reserved for its deployment. So, we developed this voice bot application using the Dialogflow conversational framework and deployed as Google Cloud Functions for Firebase. Our load testing results shows that bot performs acceptably well under spike and soak tests also. Though conceptually, we developed our voice bot for visually impaired students but we highlight that it can be used for all students in current scenario of COVID pandemic where all students can give submit their feedback through online mode using this serverless cloud application.

References

[1] "martin Fowler Serverless." [Online]. Available: https://martinfowler.com/articles/serverless.html. Access on 15 July 2020

[2] J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: the prospect of serverless scientific computing and HPC," *Commun. Comput. Inf. Sci.*, vol. 796, pp. 154–168, 2018, doi: 10.1007/978-3-319-73353-1_11.

[3] "AWs lambda." [Online]. Available: https://aws.amazon.com/lambda/. Access on 10 July 2020

[4] "Azure." [Online]. Available: https://azure.microsoft.com/en-in/. Access on 10 July 2020
[5] "GCF." [Online]. Available: https://cloud.google.com/functions . Access on 10 July 2020
[6] "IBM." [Online]. Available: https://cloud.ibm.com/functions. Access on 10 July 2020
[7] J. Kuhlenkamp, S. Werner, M. C. Borges, D. Ernst, and D. Wenzel, "Benchmarking elasticity of FaaS platforms as a foundation for objective-driven design of serverless applications," *Proc. ACM Symp. Appl. Comput.*, pp. 1576–1585, 2020, doi: 10.1145/3341105.3373948.
[8] "why-most-chatbots-fail-1c085b74d6ad @ chatbotsmagazine.com." https://chatbotsmagazine.com/why-most-chatbots-fail-1c085b74d6ad.
[9] "The 5 Best Use Cases for the Serverless Beginner" https://epsagon.com/development/the-5-best-use-cases-for-the-serverless-beginner/
[10] G. Adzic and R. Chatley, "Serverless computing: economic and architectural impact," pp. 884–889, 2017, doi: 10.1145/3106237.3117767.
[11] A. Eivy, "Be Wary of the Economics of 'Serverless' Cloud Computing," *IEEE Cloud Comput.*, vol. 4, no. 2, pp. 6–12, 2017, doi: 10.1109/MCC.2017.32.
[12] "AWS Vs Google Cloud – Detailed Comparison" https://intellipaat.com/blog/aws-vs-google-cloud/ .
[13] "canalys-worldwide-cloud-infrastructure-Q4-2019-and-full-year-2019 @ www.canalys.com." .
[14] "Pricing @ Cloud.Google.Com." Access on 15 July 2020
[15] "compute @ cloud.google.com." Access on 15 July 2020
[16] "docs @ cloud.google.com." Access on 15 July 2020
[17] "index @ devcenter.heroku.com." Access on 20 July 2020
[18] X. Luo, S. Tong, Z. Fang, and Z. Qu, "Frontiers: Machines vs. humans: The impact of artificial intelligence chatbot disclosure on customer purchases," *Mark. Sci.*, vol. 38, no. 6, pp. 937–947, 2019.
[19] I. V Serban *et al.*, "A deep reinforcement learning chatbot," *arXiv Prepr. arXiv1709.02349*, 2017.
[20] A. Xu, Z. Liu, Y. Guo, V. Sinha, and R. Akkiraju, "A new chatbot for customer service on social media," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 3506–3510.
[21] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," in *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, 2016, pp. 1–4.
[22] B. R. Ranoliya, N. Raghuwanshi, and S. Singh, "Chatbot for university related FAQs," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 1525–1530.
[23] M. N. Kumar, P. C. L. Chandar, A. V. Prasad, and K. Sumangali, "Android based educational Chatbot for visually impaired people," in

2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2016, pp. 1–4.

[24] S. M. Felix, S. Kumar, and A. Veeramuthu, "A smart personal AI assistant for visually impaired people," in *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2018, pp. 1245–1250.

[25] O. Metatla, A. Oldfield, T. Ahmed, A. Vafeas, and S. Miglani, "Voice user interfaces in schools: Co-designing for inclusion with visually-impaired and sighted pupils," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–15.

[26] J. Lehvä, N. Mäkitalo, M. T. Case, and S. Building, "This work can be found from : To cite this work , use : Authors ' preprint version bellow . Case Study : Building a Serverless Messenger Chatbot," 2018.

[27] "https://aws.amazon.com/lex/" .

[28] "what-is-luis @ docs.microsoft.com." Accessed on 22 July 2020.

[29] "index @ serverlesscalc.com." Accessed on 22 July 2020.

[30] "calculator @ cloud.google.com." Accessed on 10 July 2020.

[31] "machine-types @ cloud.google.com." Accessed on 18 July 2020.

[32] U. Bharti, D. Bajaj, H. Batra, S. Lalit, S. Lalit, and A. Gangwani, "Medbot: Conversational Artificial Intelligence Powered Chatbot for Delivering Tele-Health after COVID-19," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, 2020, pp. 870–875.

[33] "firestore @ firebase.google.com." Accessed on 10 July 2020 .

[34] "testing-best-practices @ developers.google.com." Accessed on 10 July 2020.

[35] Apache Bench, "ab-Apache HTTP server benchmarking tool." Accessed on 10 July 2020