Engineering Voice Activated Reports Leveraging Artificial Intelligence

Stefon Williams, Caleb Crickette, Thomas Barton, Sohail Sobhani, Geoff Dean
Abdul Kamara, Eugene Kim, Anthony Cedeno, Jean-nae Dedrick, Vincent Leung, Ivy Pham
Nicholas Ballo

SWEN 670 Software Engineering Project
University of Maryland Global Campus
1616 McCormick Dr, Largo, MD 20774, United States

March 29, 2021

Keywords: *Artificial Intelligence, Machine Learning, Dialogflow, Software Engineering, Google Cloud, Voice-to-Speech, Go, SPA, Web Application, Concurrency, Algorithm, RESTful API, Webservices, Google Assistant, DevSecOps, CI/CD pipeline, continuous integration, continuous deployment, containerization, orchestration, Kubernetes, remote programming, distributed computing, open source*

Abstract

A team at of the University of Maryland Global Campus pursuing a Masters of Information Technology: Software Engineering was assigned the project to build an open source application which leverages artificial intelligence to construct reports using speech to text translation. The resulting outcome was an application which used Google's Dialogflow as the main artificial intelligence service to power a webhook service which fills out custom forms utilizing Google Docs as the main form builder. The design and overall engineering approach consisted of building the most efficient tool in the quickest amount of time within the Agile Framework. Working in a global team, the team managed to build a repsonsive SPA web application, client UI, and webhook middleware msystem using VUE and Go. The project can also be used as a middleware for applications which want to make their own UI and need to make use of automated report filling and speech to text matching.

1 Introduction

The University of Maryland Global Campuss Masters degree program for Information Technology Software Engineering Capstone course offers a course of study in the form of a software engineering project utilizing the sessions cohort to form software development teams to design and implement a software application of choice. For the Spring 2021 session, the project consisted of creating an artificial intelligence software utilizing Dialogflow, a cloud service API hosted by Google, to create a mobile device application. There is an ever-present desire to save time when doing tasks, and the problem that our team decided to address was saving professional's time while interacting with clients by finding a way to automatically capture data that a professional's voice, and organize it within a form. This would allow the professional to spend less time filling out menial forms and more time engaging with their clients. The application would listen to a service professionals voice conversation with an individual in real time, process said conversation to extract relevant information from what the professional says, and automatically generate a report based on a pre-defined template. Such conversations would take place a professional and their client such as first responders, medical, and health professionals and the , but could also extend to any service that have some type of registration or form process. The components required were a processing engine using Dialogflow, a corresponding REST API, and a mobile interface to present to the user.

1.1 System Architecture

When the first version of the architecture was established, there were three overarching components which would have some overlap. The front-end mobile UI was an obvious first component as every application needs to interface with a user, even more so with this project as there are important functions that will need to be fulfilled by a nonverbal interface. Following that there were discussions how the back end would function, and it was determined that there would be two components to the backend with some overlap. The application would implement the Google Dialogflow service that would interpret the users input and organize it into data, this would then need to be received by a web server that would handle the finished documents. The Dialogflow service was also to have a web API which put the data into a database, and then the web servers would pull the data and put it into the document.

A revelation occurred midway through the project, in which it was determined that Dialogflow can take information it captures and intsert it directly into a Google Docs file, and the Google docs file can be read with specific markers to allow for the creation of fields. This greatly simplified the overall design of the Form Scriber application as the database was no longer necessary, the document management can be handled within Google Docs itself, and authentication can be handled with the

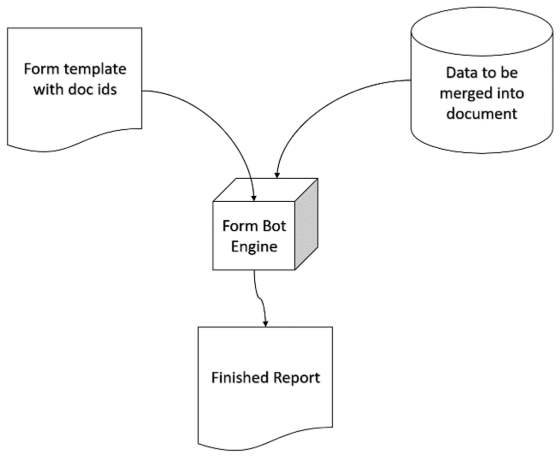


Figure 1: Transformation occurs between the data source (end-user voice) to the Google Doc template which has fields which are converted automatically into DIalogflow intents through a dynamic intent creator from the backend webservices.

Googles account environment. This also allowed for the web team to be dissolved and the teammembers redistributed to the mobile and Dialogflow teams. Figure 1 illustrates the concept behind enriching the google doc template with the data acquired from the user.
In order to accomplish the task of handling dynamic intents a dynamic intent generator function was created, which processes the template beforehand by fetching a url which the user gives over to the chatbot, see figure 2. By using unique form-ids, the backend server is able to process and interrogate the Google docs. When this was first done, the amount of data fetched by the Google Doc even for a small sized form template was over 10,000 lines of JSON. Considering this, the query had to be optimized. Luckily, through parameterized API querying and clever usage of tables in the google doc template, this size was able to be reduced in size to less than 100 on average. This resulted in processing all intents from the google doc in less than 1 second. As Googles webhook response requires responses under 10 seconds and considering the user-experience should really feel like they are

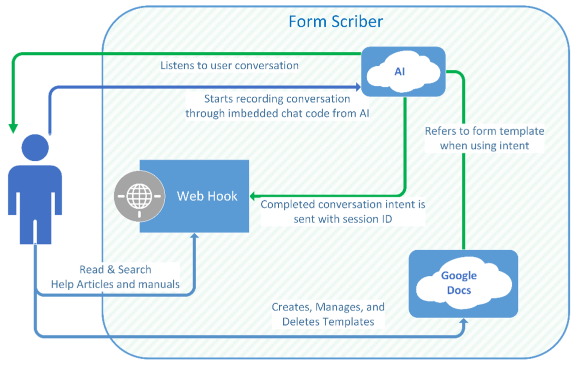

Form Scriber
Listens to user conversation
AI
Starts recording conversation through imbedded chat code from AI
Refers to form template when using intent
Web Hook
Completed conversation intent is sent with session ID
Read & Search Help Articles and manuals
Google Docs
Creates, Manages, and Deletes Templates

Figure 2: Illustration of general architecture components within the framework.

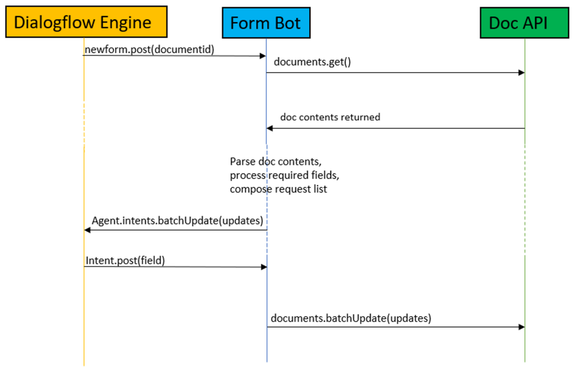

Dialogflow Engine
Form Bot
Doc API
newform.post(documentid)
documents.get()
doc contents returned
Parse doc contents, process required fields, compose request list
Agent.intents.batchUpdate(updates)
Intent.post(field)
documents.batchUpdate(updates)

Figure 3: UML sequence diagram of the RESTful API endpoints.

talking to someone, performance was a critical factor. The more performance we could squeeze the better.

Once the fields have been extracted then it was just a matter of carefully designing a generic intent generator, which can work for our purpose. This consisted of creating several contexts, such as the google doc url, session, id and client id which would be passed throughout the session. This meant that our backend service could remain stateless without the need of any database and only use very limited memory resulting in optimized performance and having the google docs and google Dialogflow service do session and state management or as Martin Fowler would say, smart endpoints, dumb pipes (Microservices guide, n.d.). This process was encapsulated in its own method with utilized a Go routine that runs concurrently. While it is running, we already send the webhook response with a customized message back to the end user. In the background, the dynamic intent generator is performing a batch update creating any new intents in the Dialogflow agent. If an intent already exists, it will not be made. The beauty of this design is that everyone can make use of all the different intents since each intent is managed in its own state and session directly with their form and reports. So, if two people have both chosen a field called

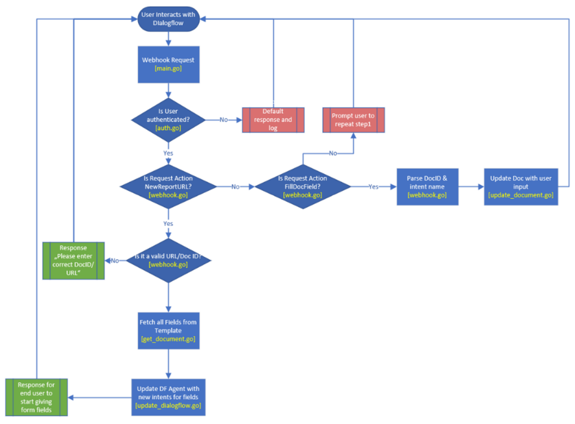
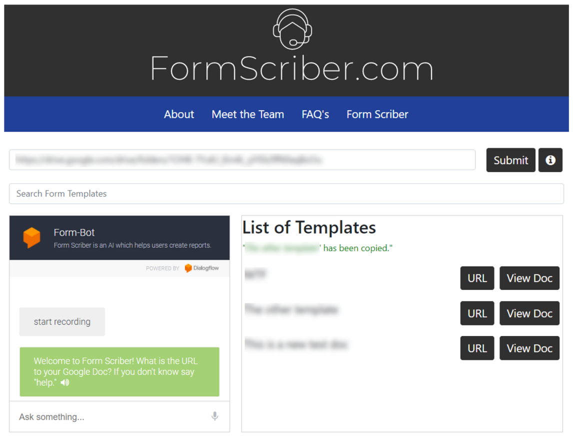

User Interacts with Dialogflow
Webhook Request [main.go]
Is User authenticated? [auth.go]
No
Default response and log
Prompt user to repeat step1
Yes
No
Is Request Action NewReportURL? [webhook.go]
No
Is Request Action FillDocField? [webhook.go]
Yes
Parse DocID & intent name [webhook.go]
Update Doc with user input [update_document.go]
Yes
Response „Please enter correct DocID/ URL"
No
Is it a valid URL/Doc ID? [webhook.go]
Fetch all Fields from Template [get_document.go]
Response for end user to start giving form fields
Update DF Agent with new intents for fields [update_dialogflow.go]

Figure 4: Form Scriber engine workflow

date they can both use it. Meaning our Dialogflow agent gets updated and trained in the background and works faster the more forms people make with it. Figure 3 illustrates the sequence diagram of the overall process. Figure 4 illustrates the overall of the program.

In order to interrogate the Google Drive and Google doc without using a SQL database and maintaining state in our backend system, we wrote a small function which generates and signs its own JSON Web Token to send to Google Oauth2 server for creating a valid token to access the parts of these systems. This has several advantages in that user only needs to share their google drive with our service



Form Scriber is an AI which helps users create reports.

Welcome to Form Scriber! What is the URL to your Google Doc? If you don't know say "help."

List of Templates

"' has been copied."

Figure 5: Screenshot of the Desktop housing the user guides and Form Scriber client implementation using Dialogflow Messenger and Google Drive integration.

account once. This shared access gets inherited in all their documents in that folder on their drive and they can choose to revoke our service account at any time. If revoked and they try to access, our backend will inform the user accordingly that it could not access it and direct them to our website for further guides and information. In addition, we created a front-end website which is a Single Page Application utilizing the VUE framework. The purpose of the website it two-fold. On the one hand it serves to host all the user guides, FAQ, and documentation to help users understand how to prepare the documentation. During development we then had the idea to extend the Dialogflow client directly into the website which is responsive meaning that it can work on both mobile and desktop devices. In the end we truly had an all-round application capable of delivering an end-to-end report-making solution. Figure 5 illustrates what the page looks like to a desktop user.

1.2 Software Analysis

1.2.1 Software Positives

- The benefit of capturing the url from the user is that it gets stored in Dialogflow as a context within the complete conversation. That makes it very lightweight and scalable so we always know what we are working with.
- The dynamic nature of capturing intents that are inserted into a form makes the application flexible for what forms can be used.
- Simple field markup allows for anyone to be able to make fields easily.
- the name of the first author should be repeated in the upper right-hand corner of each page.

1.2.2 Software Negatives

- Currently has decentralized management, documents have to be accessed outside of the overall application in Google Docs.
- Voice recognition accuracy is dependent on Googles voice recognition software, which while good is not 100

1.2.3 Lessons Learned

- We first tried traditional method of create a rest web services and web site separated from the app and use it as data handler between the mobile app and dialogflow. As we developed the architecture we discovered that it could even be further simplified into a centralized Go application.
- We learned that to produce quality software in a globally distributed team, an experienced global project manager is paramount. Without someone to keep teams centrally organized and on track it is very easy for other teams not to know what the other is doing. This can result in wasted effort and miscommunication such as the other team might assume something else about the other teams architecture. This is a very challenging role and should be someone with good programming experience and knowledge.

- Flexible scheduling was important to fully implement due to bumps and hiccups in parts of the development process.

2 DevSecOps

The multiple components and integration required, presented an opportunity to incorporate a DevSecOps (DSO) team to create a continuous integration and continuous deployment pipeline utilizing state of the art trends such as application containerization and container orchestration. The capstone practicum was therefore also a testing ground to include a development and operations (DevOps) culture and associated tools and infrastructure to be used by the Dialogflow component. DevOps practices represent a fundamental shift in culture from a traditional software development lifecycle where multiple departments previously siloed in responsibilities adopt a system of close collaboration using new tools and ways of working to release high quality products in faster and more frequent cycles to customers (Willis, Debois, Humble, Kim, 2016). The emphasis on this model is the ability to be agile in the face of ever-changing requirements and enable faster feedback loops from the products end users. As a result of release time improvements and infrastructure robustness, an organization or project can be more efficient, reliable, and productive, leading to increased growth and profitability (Willis, et al., 2016). It is no wonder that with all these benefits, more and more teams are adopting development operations, growing to 74 percent of software organizations (Weins, 2016). DevOps continues to be a transformative trend in the software development industry as it promises effective organizational culture, structure, and high-quality software. We plan to explore and analyze the effects of integrating the DSO project into FormScriber Dialogflow in order to inform future efforts.

2.1 CI/CD

The DSO project was to create and facilitate the usage of a DevSecOps infrastructure that automates the building, testing, securing, and deploying of the development teams application continuously. A continuous integration (CI) pipeline with a centralized code repository was provided to enable this automation and provide static code analysis. A continuous delivery (CD) pipeline was also conceptualized to build and deploy said application. The foundation of the DevSecOps infrastructure consisted of the source code repository and GitHub was selected as the tool of choice to integrate with existing and new infrastructure components. Using pull requests, the team was able to see in real-time automated scans and checks that were conducted by SonarCloud, a code quality tool, for the code being pushed. Scans provided feedback regarding compliance to coding best practices as well as security best practices. Any bugs or vulnerabilities were identified and flagged during the scan upon the pull request, and the status was updated immediately to the repository before a merge was possible. This ensured a higher-quality end product and provided a fast feedback cycle.

The Dialogflow development team was able to take advantage of a DevOps pipeline created in Azure DevOps to automatically build and monitor the status of the codebase. The pipeline integrated with GitHub and provided tasks by which to pull the code from the repository, install dependencies, and execute application build commands. This was done on each pull request and merge into the main repository branch. Having an automated solution integrate code from multiple contributors allowed the team to have higher confidence in the integrity of the product and that potential issues related to environment dependencies could be more removed.

2.2 Containerization

To improve upon this, a parallel research and development effort into a DevSecOps framework, Advance

Development Factory (ADF), was completed for the FormScriber Dialogflow component to support the Go programming language. ADF provided a means to reduce the difficulties that may be present when installing, configuring, or updating dependencies upon which a development environment requires to build, test, and deploy an application. This was possible through the use of containerization to create a Docker image that abstracted out an environment with all required dependencies to build and deploy Dialogflow components.

An official Docker image for Golang was chosen, based on an existing Debian Linux distribution. Within this image, the DSO project defined, via a Dockerfile, dependencies to install, such as git, make, which, wget, as well as others specific to the project such as helm, Azure CLI, Docker engine, and kubectl. Separately, an Azure configuration file defined a pipeline that would 1) automatically build this new Docker image with the additional dependencies, 2) log into a container repository at docker.io, and 3) push the image to said repository upon code changes to the ADF framework. This extended a previous version of ADF that relied on a Makefile to manually do these tasks from a local command line. The Makefile was not removed in order to provide backwards compatibility and additional build and deploy options.

The resulting new ADF-Golang-Dialogflow Docker image became the base image that could be integrated into the Dialogflow project. The DSO project conceptualized the usage of this framework by utilizing an existing Makefile from the Dialogflow development team as well as the deployment guide to add tasks related to the building, creation of artifacts, and deployment of the application from within the image. By executing the commands via make, ADF would enable the tasks of pulling the latest ADF Docker image from the Docker Hub repository, running the container, and then from within the container, creating the various Azure and Kubernetes resources and application Docker image, and lastly deploying to Azure Kubernetes Service (AKS). To wit, under the ADF framework, all that would be required for a user to accomplish all these tasks would be to install git and Docker on their local command line.

2.3 ADF and Beyond

Due to the constraints of the given short timeframe, a release pipeline was realized for the base ADF framework. The previous build pipeline for ADF created the Docker image and release pipeline configuration files as artifacts that were utilized for the release pipeline. The release pipeline completed the CI/CD pipeline by deploying the ADF Docker image to an active AKS cluster that was pre-configured by the DSO team. The success of this pipeline fully realized the end-to-end build and deploy automation that was a goal by the team at project inception. The resulting infrastructure and pipeline configurations for ADF will inform future development cycles by acting as a working template and proof-of-concept.

This infrastructure could be customized and integrated further into the Dialogflow project to provide additional facilitation for deployments. The DevSecOps integration into the FormScriber project allowed for the Dialogflow team to harness industry-accepted practices related to IT infrastructure, Agile methodology alignment, and take advantage of current state-of-the-art trends in software development. This provided invaluable feedback to development team members and stakeholders on the status and health of the development lifecycle during every milestone, as well as provide the necessary background and platform for future improvements.

3 Next Steps

While we were able to meet most of the requirements necessary to capture the voice and automatically fill out any type of form the user wishes to create, there are few ways this project could be improved and taken further.

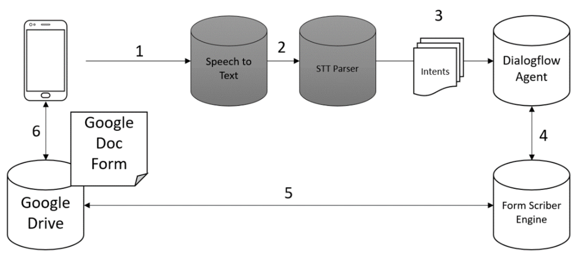


Figure 6: How the Form Scriber engine can be used as a core middleware for other implementations, specifically one which records an entire conversation at once and then feeds this information into the form scriber engine.

Due to the restrictions of Dialogflow and Google Assistant it is not feasible to leave a mic open and record the entire conversation since Dialogflow agents always need an intent. To get around this another system would need to be established such as a voice to text app which can rcord everything and then process this data into pieces or sentences. Once the entire recording is broken up the Dialogflow APIs can be used to send this data over and use our existing application and structure which would automatically assemble any matching intents to the users reports. Such a system would look like the following in Figure 6.

As can be seen only two extra parts need to be created and no changes are required in the Form Scriber Engine, which can process any form and handle any dynamic intents. The idea here is that instead of conversating with the Dialogflow agent, a user can just record an entire conversation non-stop and then asynchronously, when the conversation is over it can produce a report. A possibility is that an email could be given by the user so that it can be emailed back to the user since it will not be in real-time.

This is one of the disadvantages of going this route namely, that it is not happening in real time, but will be an asynchronous process once the recording is finished. Also, if such a task is undertaken, engineers will need to deal with the possibility of very large files which might be a nuisance on mobile devices with limited bandwidth since the recording is on the entire time and will then need to be processed later. There must be special attention given to the processes and performance of the backend systems in such a design. Currently, the system occurs in real-time, is highly accurate in placing the speech to the correct field, very fast, and lightweight. The main question is then, is the performance and potential accuracy loss due to feeding the intents automatically versus in a dialogue, worth having the ability to leave the recording on and capturing the entire conversation? It would be interesting to take this further and to benchmark accuracy and performance against the current model. Developers could then compare the pros and cons against the latter.

4 Conclusion

We were able to deliver the requirements of providing a tool which uses artificial intelligence, capable of generating any possible type of report using only the users voice within the margin of acceptance of using Google Dialogflow, 7 weekly Sprints, and no budget. There are improvements that can be made if one would like a tool which records everything in one large recording without needing to engage in a dialogue with the Form Scriber tool. We were of the opinion that due to the project constraints, limited time and budget we could not implement nor have the time to carry this additional requirement out as it would require 2 separate modules and additional backend programming. Instead, due to the limitations, we focused on building the best possible system which worked which could be used as a stand-alone or used as a core to another application. We feel this approach was the most productive and allowed us to scale our resources effectively by prioritizing our use-cases based on our Agile framework.

5 Acknowledgement

We would like to extend our appreciation and gratitude to our professor Dr. Mir Assadullah and course mentors: Roy Gordon, and Johnny Lockhart. For without all their feedback, advice, and guidance this project would have not been possible.

6 References

[1] Microservices guide. (n.d.). Retrieved March 23, 2021, from https://www.martinfowler.com/microservices.

[2] Weins, K. (2016).New DevOps trends:2016 state of the cloud survey. Retrieved from https://www.rightscale.com/blog/cloud-industry-insights/new-devops-trends-2016-state-cloud-survey.

[3] Willis, J., Debois, P., Humble, J., Kim, G. (2016).The devops handbook.Retrieved from http://www.safaribooksonline.com.