Help Desk Helper

by

Aaron Sellmeyer, Turin Auguste & Michael Meier

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

Aaron Sellmeyer	4/12/2021
Turin Auguste	4/12/2021
Michael Meier	4/12/2021
Faculty Advisor: Ryan Moore	4/12/2021

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2021

Contents

List of Tables & Figures

Tables

Figures

Abstract

Since February 2020, tickets coming in over channels like WhatsApp, Facebook Messenger, direct messages over Twitter, and SMS rose nearly 50 percent. While this is convenient for the user, this could result in tickets not being generated for the help desk and may be negatively impacting their tracking metrics. Our solution integrates itself within multiple commonly used messaging and support desk platforms. This provides end users with easy solutions for their common problems and granting them an easy way to make a support ticket should they require additional support from their help desk. By using readily available technology, we are bringing an easy-to-use solution to those in need of support in a way that is convenient for the end user and easy for the support desk to track. Our Solution provides an easily accessible chatbot that provides end users with quick level one support.

Introduction

Problem Statement

With many companies recently shifting to working from home, service desks at these companies around the world have seen an increase in support calls. Compared to the same time last year, support ticket volume is up 16% as seen in figure 1 (zendesk.com). Also, with lots of people working from home, they are no longer able to visit their help desk in person when experiencing an issue. This has caused those users having an issue to turn to chat platforms such as Slack, Teams and Facebook Messenger to seek out support for their issues. Since February 2020, tickets coming in over channels like WhatsApp, Facebook Messenger, direct messages over Twitter, and SMS rose nearly 50 percent (zendesk.com). While this may be convenient for the user, this could result in tickets not being generated for the help desk and may be negatively impacting their tracking metrics.

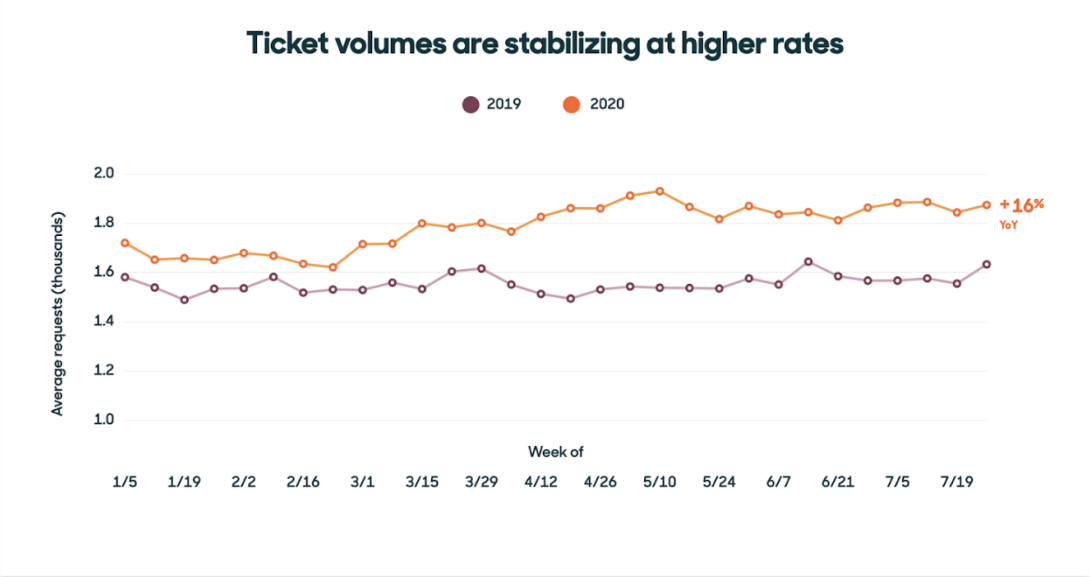

Figure 1 - Increase of ticket volume from zendesk.com

Solution

Help Desk Helper decreases the total amount of time users spend waiting for support and alleviates some of the stress and increased workload of help desk workers because of Covid-19. Our extension integrates itself within multiple commonly used messaging, communication, and support desk platforms. This service provides a chatbot aimed to tackle basic help desk support for employees experiencing issues. If an issue is not sufficiently resolved during the initial exchange, then the bot will create a ticket and address it to the correct upper echelon of support. Additionally, the bot can be contacted to communicate known outages with end users, answer basic questions and generally point the user in the right direction in terms of computer support. This solution helps streamline the process of creating tickets by grabbing required information for a ticket to be generated including the users name, a brief problem description and steps taken. We aim to do this through the creation of a QnA bot hosted in Amazon Web Service (AWS) and connect to many different messaging software (Slack, Teams, etc.) that will call our cloud hosted Knowledge Base based off of the users input and provide troubleshooting steps to the user. Should the issue not be resolved by the bot or if the user is not satisfied, the bot will create a ticket through information gathered in the interaction with the user and generate a ticket to be sent to the help desk through an API call to existing support infrastructure.

Project Goals

The main goal for our project was to create a level 1 chatbot hosted on a cloud service provider that can be accessed over multiple communication channels. The intention for

the chatbot is to solve the most basic technical issues users face to free up time for help desk personal to focus on more advanced issues. However, if the chatbot cannot solve a user's issue it will gather any information it can and forward it to our ticketing server to automatically open a ticket. Other goals for the project include having the chatbot broadcast known outages and provide other tips and tricks to users to help them stay on top of their technology and prevent future issues.

Overview

The remainder of this report describes in detail some of the more granular parts of the project. First, we explain the "big picture" of the overall project and experiences that led us to come up with the idea. Next, we break down the goals we achieved, and the methodology used to reach those goals. After that, we show our supporting documents for the project such as the User Profiles, Use Case Diagrams, timeline, and budget. Finally, we describe problems we encountered throughout the project and our solution to tackle them.

Discussion

Project Concept

The team initially formed when Michael and Aaron met during one of the Senior Design orientation meetings. Turin joined after responding to a discussion board post created by Michael asking if anyone else was interested in joining the team.

The inspiration behind the project came from Aaron who recalled an issue he faced during his time working at a help desk. The issue was that sometimes users would message him directly asking for technical assistance instead of following the normal procedure and opening a ticket. Users would also commonly post in very large group chats, tagging every member of the group seeking assistance. Sometimes their questions were issues that could easily be resolved by contacting the level one help desk and other times, they were known outages. Users would sometimes state that they did not know how to contact the proper channels for their issues while other times, they said that it was too much of a hassle. With our idea, our bot can be contacted directly by those same end users that were sometimes tagging thousands of people regarding a known outage or an easily fixed problem had they just contacted the correct people. This results in a win for the end users, who will now have a convenient and direct way of seeking support and a win for the service desks as the bot will create tickets to be sent to the appropriate team(s).

Design Objectives

Creating a fully functioning level 1 support chatbot that can be accessed on multiple communication platforms was the primary goal of the project. We want users to feel comfortable using familiar applications like Slack and Microsoft Teams to communicate with the chatbot. Since most companies already have these chats setup, it will be easy to integrate our chatbot to them.

Standing up our own ticketing server was another big goal for the project. The initial plan was to use our own hardware to host the server however in the interest of convenience we switched to a cloud hosted solution.

Creating the link between the chatbot and both the ticketing server and communication platforms was the last main goal for the project and arguably the most important. AWS made it easy to configure the connection between the chatbot and the communication platforms but the link between the chatbot and the ticketing server was a little more difficult.

Methodology & Technical Approach

We knew right away our chatbot was going to be hosted using one of the major cloud computing providers (AWS, Azure or Google Cloud) so our first task was to experiment with each of them. After doing some tests and research, we found that AWS would be the optimal cloud provider for this project because not only was it extremely user friendly, but it has also built-in services tailored to what we are looking for (Amazon Lex and

Amazon Lambda). Lex is used to create the actual chatbot and allows us to code what are called “Intents” that the user can interact with. Each intent is an issue a user could face. For example, the keyboard issue intent will guide users through various troubleshooting steps if their keyboard is not functioning properly.

Lambda is used to achieve our link between the chatbot and the ticketing server. Think of Lambda as more back-end coding which can both store and forward variables it picks up in the chatlogs. These variables include things like the username, email address and a brief explanation of the issue. With Lambda, we created a Lambda function to make the connection to our ticketing server.

As for the ticketing server, we decided to use the open-source software called osTicket. Not only does osTicket fit within our budget, but it also can handle HTTP API requests send from our Lambda function. After further development, we hope to expand our solution to more ticketing solutions such as Spiceworks and Jira Service Desk.

User Profile

Tables 1-3 display our User Profiles. Help Desk Helper will be used by three types of users. The first user group will be made up of people who are seeking technical assistance (end users). The second user type will be the administrators who will monitor the application and ensure it is working properly post release. The last user group will be the

support desk personnel who will monitor the ticketing system and periodically add more functionality to the chat bot.

End Users
Application: Microsoft Teams, Slack, Telegram, Twitter and other common communication platforms
Potential Users: All employees in the company
Software and Interface Experience: The user should be familiar with common communication software and know how to navigate to the help desk channel for their respective communication software.
Task Experience: The user should have knowledge of their computer and/or software they are requesting help on. They will need enough knowledge to properly describe the issue they are experiencing.
Frequency of Use: This will be based on the company's history of help desk tickets their end users submit.
UX Design Requirements that the Profile Suggests: The bot should be easy to access and navigate by the end user. The bot should use friendly and non-technical terms so all end users can easily interact with it.

Table 1 - End Users

Administrators
Application: Microsoft Teams, Slack, Telegram, Twitter and other common communication platforms. AWS, Azure, Google Cloud and other common cloud providers. Open-Source Ticketing Systems such as Spiceworks, Jira Service Desk and osTicket.
Potential Users: Systems administrator and developers
Software and Interface Experience: The admins and developers should be familiar with all of the integrated communication applications. They need to have intimate knowledge of their cloud platform and how it interacts with their chat service and ticketing system in order to maintain the service.
Task Experience: This user group will be responsible for monitoring uptime and resource usage as well as updating the application. They will also ensure the bot is functioning properly and review user feedback.
Frequency of Use: Once the extension is created and implemented, the user will only interact with the extension as needed. The user will correct any issues that arise post-launch and update features based on end users comments.
UX Design Requirements that the Profile Suggests:

The bot itself should be easily configurable allowing the admins to make changes when needed. The hosted applications should be in the network to allow the admins access to it.

Table 2 – Administrators

Support Desk Personal
Application: Open source ticketing software and applications determined to be in their scope of support.
Potential Users: Personnel on various support teams that will maintain the knowledge base to be referenced bot the bot. This user group will also handle ticket escalations from the bot.
Software and Interface Experience: This group should be familiar with their support infrastructure and have a working knowledge of applications in their support scope. If they update support documentation for a certain application, they should be able to replicate that in the knowledge base for the bot.
Task Experience: Handling ticket escalations from the bot while also ensuring that the bot has the most up to date knowledge of support techniques that can be easily explained to end users. Support tickets created by the bot will be closed at this level.
Frequency of Use: In addition to normal ticket escalating measures, this group will primarily work off of tickets that are created and escalated by the bot.

UX Design Requirements that the Profile Suggests:
The bot needs to create accurate tickets with good problem descriptions and user information that the help desk team can use to contact the user for further troubleshooting.

Table 3 - Support Desk Personal

Use Case Diagram

Figure 2 shows our Use Case Diagram. As seen below, the end user interacts with the communication tool by initiating the bot, asking for support, and providing feedback. Think of this whole interaction happening in a Slack channel or Facebook Messenger instance. The end user is using familiar programs to get the help they need.

The administrator will interact with both the chatbot and the communication tool and will essentially be monitoring the health metrics of both systems. Think of this as more of the backend support.

Finally, the support desk personnel will be interacting with the ticketing system to resolve any tickets sent from the chatbot. Again, the support desk personnel will be using familiar programs to provide support and will not have to interact with the bot at all.

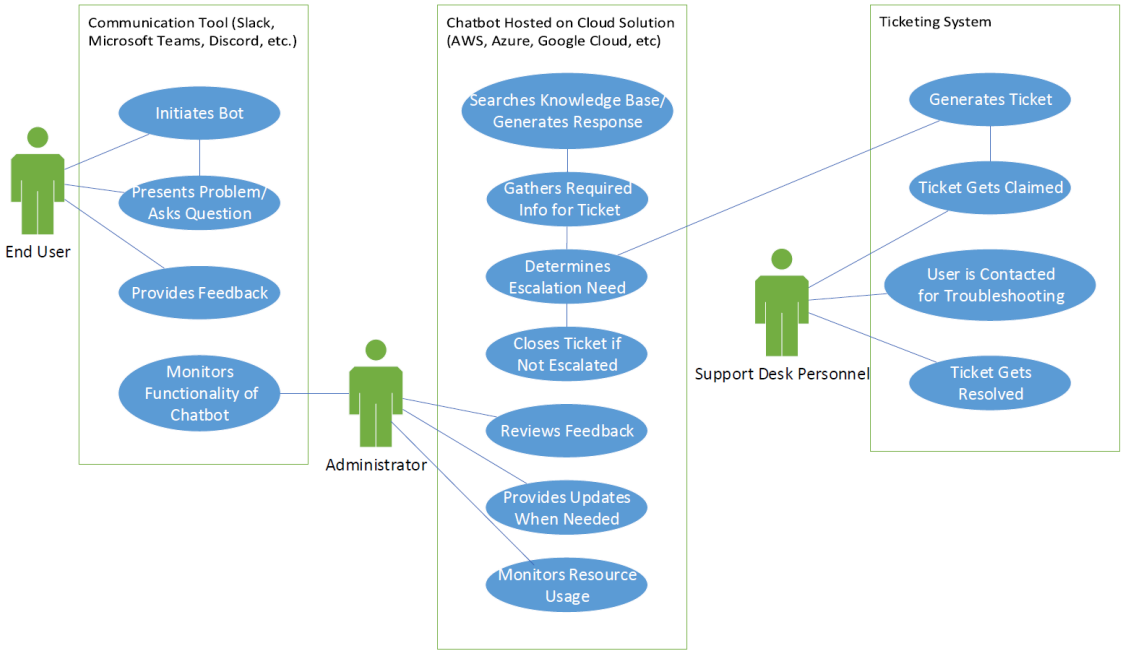

Figure 2 - Use Case Diagram

Technical Architecture

Figure 3 shows the technical architecture for HelpDesk Helper. There are three major components of the project: the EC2 instance, the Lex and Lambda chat interface, and the supported communication tools.

Figure 3 - Technical Architecture Chart

The EC2 instance is where the support tickets are generated and stored. This is done by using an AWS hosted Ubuntu server running Nginx as the webserver which is hosting the osTicket server. Information for the tickets is stored in a MySQL database.

AWS Lex is where the bot conducts the conversations with the end users. Lex has the supported issues which are called intents and where the user information is collected which are called slots. Lex then passes that information to the Lambda function for troubleshooting steps.

AWS Lambda is the brain of the bot. Each supported intent has its own function, which is written in node.js and is where the troubleshooting steps occur. The Lambda function uses a switch case to walk the user through the troubleshooting steps and will return information based off user input. Figure 4 is an example of how the switch case works.

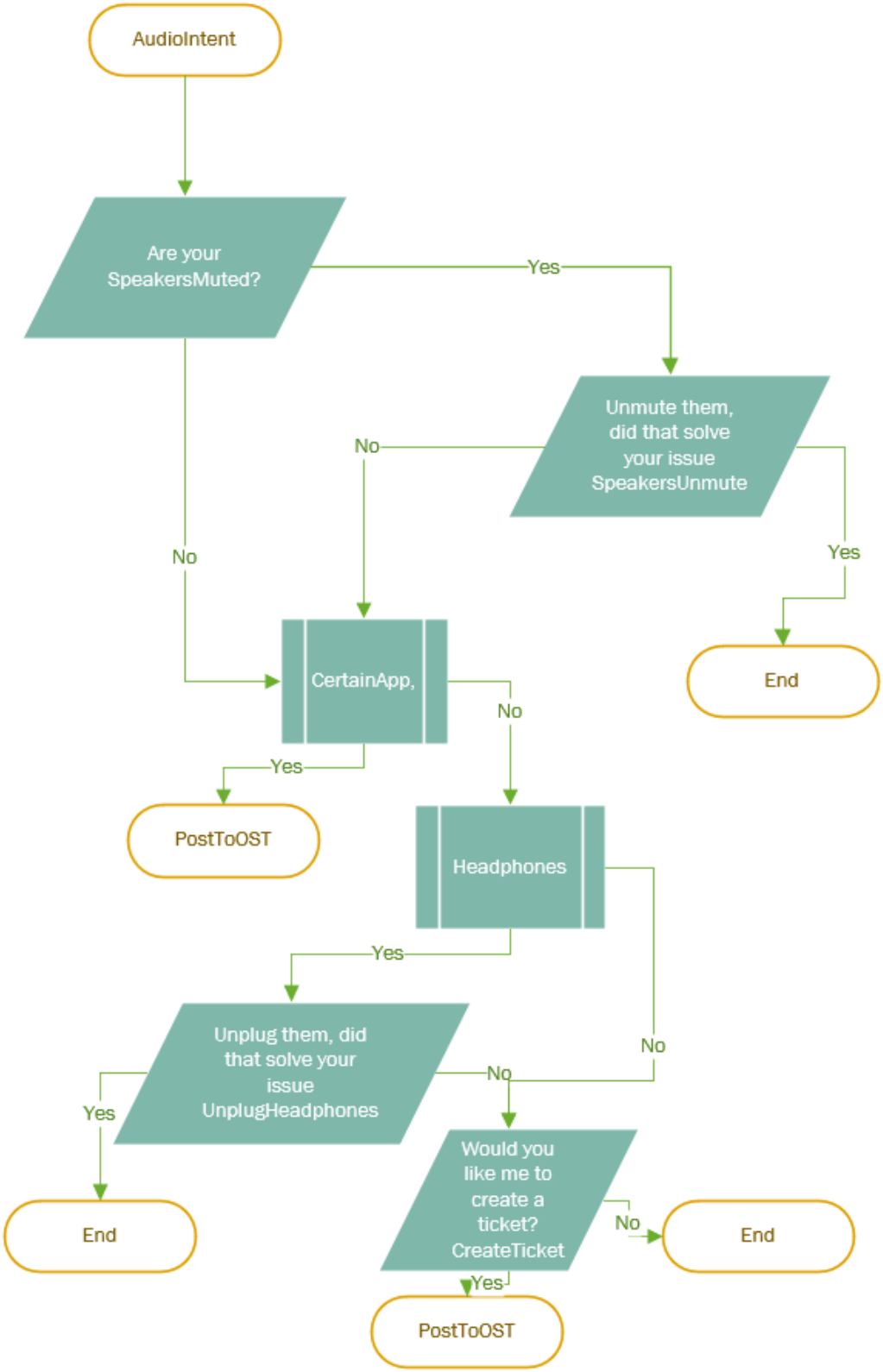

Figure 4 - Lambda Switch Case

The supported communication tools are the chat applications that are tied into Lex by using webhooks. We tested Slack, Facebook Messenger, and Kik in development of our project. We expect that more applications could be added based on the needs of the company implementing Helpdesk Helper.

Testing

Our testing methodology breakdown looks like the following:

- Functional Methods Unit Testing
    - Integration Testing
    - Acceptance Testing
- Non-functional Methods
    - Performance Testing
    - Security Testing
    - Compatibility Testing

Unit testing was done to ensure each unit of the project works correctly as a standalone section. For example, testing the Lex intents using the self-test feature provided in AWS. This was conducted on the major units of our project which includes the following:

- AWS Lex Intents
- AWS Lambda Functions
- AWS API Gateway
- osTicket Integration
- Communication platform (multiple)

System testing ensures each unit works when integrated with the rest of the units. For example, testing the integration of the Lex intents with the Lambda functions. This was done internally as we are the most familiar with the unit's functionality. The results for this testing were exactly what we expected which was everything perfectly integrating with each other.

Acceptance testing is a way to verify the target users of our project are happy with the results and can be broken down into two sections. The first is having an actual helpdesk employee interact with our osTicket server by reading tickets, identifying the issue, and providing feedback. The intention is to ensure the bot is gathering enough information from the user for a helpdesk employee to understand. The second part of acceptance testing involves having test users interact with the Lex bot. Our intention is to have users with average or below technological skills test out the bot so we can see if the bot is providing clear enough information for the user to troubleshoot on their own. Testing results for this were very positive. We used helpdesk employees from the companies we are interning at to review the information we gather for tickets and they all approved. We then began testing the chatbot on users of various technical skills. Those with more technical experience were very pleased with the directions the bot was posting however those will less technical skills requested a more friendly and detailed response from the bot.

Performance testing was conducted to ensure the Lex bot and osTicket server can handle multiple requests at once. Since our whole project was developed through cloud

computing, we can essentially increase or decrease the performance based on needs. This testing was done internally since we have access to the performance metrics of the project. Testing results for this showed exactly what we thought. We all ran the chatbot at once and saw the spike in usage on AWS but experienced no issues from the bot.

Security testing was done to verify that anything facing the Internet is hardened and that anything the Lex bot outputs has been sanitized of sensitive information. We have done this internally as we have access to the security configurations of the project. Since our group has 2 cybersecurity students, we were confident in our ability to configure security features for the osTicket instance. Testing for this included running various port scans against the public IP for the server as well as ensuring we are all using strong passwords.

Compatibility testing was done to verify our Lex bot can integrate with any communication channel. This was also done internally since we have access to the configurations for integrating the bot with communication channels. Testing results were somewhat positive for this. Amazon supports Lex integration for several communication channels which we setup. The challenge is trying to integrate Lex with something not supported by Amazon such as Telegram, Discord, Microsoft Teams, etc. Fully integrating Lex would require a lot of backend code which we simply didn't have time for.

Budget

Table 4 shows our budget. It is broken down into two main sections with the first being the software and hardware costs for the project. We tried to keep our AWS running price at $0 by utilizing as many benefits of the free tier as we could however when everything was all said and done, the cost comes out to about $20 per month. This cost includes running the EC2 instance for osTicket as well as all the computing costs for Lex and Lambda. Our price for osTicket is $0 since it is open source and our price for the communication channels used is also $0 since they are all free to use. The second section is the labor cost. We divided this up into our research phase and design phase for AWS, osTicket and the API.

Help Desk Helper 6 Month Budget				
NO.	**ITEM**	**UNIT, HOURS**	**UNIT PRICE**	**TOTAL**
Software & Hardware				
1	Amazon Web Service	1	$20 (monthly)	$60
2	osTicket	1	$0	$0
3	Communication platforms	1	$0	$0
	Subtotal			$60
Labor				
4	Research	1	$300 (flat fee)	$300
5	AWS Design & Development	1	$500 (flat fee)	$500
6	osTicket Design & Development	1	$500 (flat fee)	$500
7	API Design & Development	1	$500 (flat fee)	$500
	Subtotal			$1,800
	Total			**$1,860**

Table 4 - Budget

Project Timeline

Table 5 displays our current timeline for the project. It is broken down into 6 main sections which include Project Management and Deliverables, Research, Design, Development, Testing and Implementation.

Task Name	Duration (Days)	Start Date	End Date
1. 0 Project Management and Deliverables		8/24/20	TBD
1.1 Team Building	0	8/24/20	8/24/20
1.2 Ideas and Brainstorming	0	8/24/20	8/24/20
1.3 Fall Semester Assignment 0: Team Members & Project Name	0	8/24/20	8/24/20
1.3.1 Project Name	0	8/24/20	8/24/20
1.4 Fall Semester Assignment 1: Team Contract	7	8/24/20	9/1/20
1.4.1 Project Approval	24	8/24/20	9/17/20
1.5 Fall Semester Assignment 2: Project Abstract for Tech Expo	21	9/21/20	10/12/20

1.6 Fall Semester Assignment 3: Team Contract Resubmission	21	9/21/20	10/12/20
1.7 Fall Semester Assignment 4: User Profile	21	9/28/20	10/19/20
1.8 Fall Semester Assignment 5: Use Case Diagram	21	9/28/20	10/19/20
1.9 Fall Semester Assignment 6: Draft Report	14	10/26/20	11/9/20
1.10 Fall Semester Assignment 7: Final Fall Semester Report	19	11/9/20	11/30/20
1.11 Fall Semester Oral Presentation	21	11/9/20	11/30/20
1.11.1 Presentation Practice	21	11/9/20	11/30/20
1.12 Spring Semester Assignment 1: Testing Plan/Report	7	2/1/21	2/8/21
1.13 Spring Semester Assignment 2: Abstract	7	2/8/21	2/15/21
1.14 Spring Semester Assignment 3: Draft Tech Expo Poster	14	2/15/21	3/1/21
1.15 Spring Semester Assignment 4: Final Poster	14	3/1/21	3/15/21

1.16 Spring Semester Oral Presentation	0	3/29/21	3/29/21
1.16.1 Presentation Practice	14	3/15/21	3/29/21
1.17 Spring Semester Assignment 5: Final Presentation	7	3/29/21	4/5/21
1.18 Spring Semester Assignment 6: IT Expo Recorded Presentation	7	3/29/21	4/5/21
1.19 IT Expo	0	4/13/21	4/13/21
1.19.1 IT Expo Exhibit and Preparation	14	3/29/21	4/13/21
1.20 Spring Semester Assignment 7: Plagiarism Review for the Final Paper	7	4/5/21	4/12/21
1.21 Spring Semester Assignment 8: Final Report	7	4/5/21	4/12/21
1.22 Spring Semester Assignment 9: Final Library Copy	12	4/14/21	4/26/21
2.0 Research	24	10/5/20	10/29/20
2.1 Cloud Hosting Requirements	7	10/5/20	10/12/20

2.1.1 Research Pros and Cons of Azure	7	10/5/20	10/12/20
2.1.2 Research Pros and Cons of Google Cloud	7	10/5/20	10/12/20
2.1.3 Research Pros and Cons of AWS	7	10/5/20	10/12/20
2.2 Software Requirements	10	10/5/20	10/15/20
2.2.1 Determine Communication Tools	7	10/5/20	10/12/20
2.2.1.1 Research Microsoft Teams Compatibility with Cloud Host	7	10/5/20	10/12/20
2.2.1.2 Research Slack Compatibility with Cloud Host	7	10/5/20	10/12/20
2.2.1.3 Research Discord Compatibility with Cloud Host	7	10/5/20	10/12/20
2.2.1.4 Research WhatsApp Compatibility with Cloud Host	7	10/5/20	10/12/20
2.2.2 Determine Ticketing Software	7	10/12/20	10/19/20
2.2.2.1 Research Spiceworks Compatibility with Cloud Host	7	10/12/20	10/19/20

2.2.2.2 Research Jira Service Desk Compatibility with Cloud Host	7	10/12/20	10/19/20
2.2.2.3 Research osTicket Compatibility with Cloud Host	7	10/12/20	10/19/20
2.2.4 Identify QnA Bot Technology	4	10/12/20	10/16/20
2.2.5 Determine API Call	4	10/12/20	10/16/20
2.2.6 Research Compatibility for all Components	3	10/12/20	10/15/20
2.3 Security Requirements	7	10/19/20	10/26/20
2.3.1 Determine Access Requirements	7	10/19/20	10/26/20
2.3.2 Determine Firewall Requirements	7	10/19/20	10/26/20
2.3.3 Establish Service Roles	7	10/19/20	10/26/20
2.4 Miscellaneous Research	7	10/26/20	11/2/20
2.4.1 Identify Most Common Desktop Support Requests	7	10/26/20	11/2/20

2.4.2 Conduct Interviews with Desktop Support Teams	7	10/26/20	11/2/20
2.4.3 Conduct Interviews with Mock Stakeholders	3	10/26/20	10/29/20
2.4.4 Budget Analysis	3	10/26/20	10/29/20
3.0 Design	14	10/29/20	11/12/20
3.1 Design Cloud Environment	14	10/29/20	11/12/20
3.2 Design QnA Bot	14	10/29/20	11/12/20
3.3 Design Support Infrastructure	14	10/29/20	11/12/20
4.0 Development	18	11/12/20	11/30/20
4.1 Setup Cloud Environment	18	11/12/20	11/30/20
4.2 Create QnA Bot	18	11/12/20	11/30/20
4.3 Create knowledge base for Bot	18	11/12/20	11/30/20
4.4 Create bridge to support system	18	11/12/20	11/30/20

5.0 Testing	70	1/11/21	3/22/21
5.1 Perform Server Testing	28	1/11/21	2/8/21
5.2 Perform QnA Bot Testing	14	1/18/21	3/1/21
5.3 Performance Testing	7	1/25/21	3/1/21
5.4 Bug Fixes	21	3/1/21	3/22/21
6.0 Implementation	22	3/22/21	4/13/21
6.1 Create Models	7	3/22/21	3/29/21
6.2 Create Expo Presentation	14	3/29/21	4/13/21
6.3 Final Check on Documentation and Models	7	4/5/21	4/13/21

Table 5 - Project Timeline

Problems Encountered and Analysis of Problems Solved

Overall, our project ran smoothly throughout both semesters. Any issue we faced we worked together on solving as soon as we could to keep the project moving. The first of such issues was deciding on who our cloud service provider would be. We started to explore Azure using the free student credits that we were given by UC. Within a week,

we had used over a quarter of our credits and decided that it was not feasible to continue using Azure. This problem forced us back to the drawing board for what we consider to be a critical part of our project. We explored other cloud service providers and landed on Amazon Web Services. We found AWS free tier to be much better than Azure and we also ended up liking AWS a lot more. While we initially felt like revisiting our cloud service provider was a step back for our project, we were all extremely happy in our new direction with AWS.

The next major problem we encountered involved what to do about our osTicket server. The initial plan was to host it on our own hardware to save costs but we became hesitant after setting up the server on a home network due to all the ports needing to be opened. To solve this, we simply created an EC2 instance in AWS and hosted the server there.

Finally, creating the link between AWS and osTicket proved to be a big challenge for us. We first tried creating a custom API that would send the variables Lex picks up to osTicket for the ticket creation. After a few weeks of trial and error we went back to the drawing board and came up with the idea of using a Lambda function to send the variables. The lesson here was that AWS has extremely powerful tools like Lambda that we were not using to their full potential.

Future Improvements & Recommendations

If we were starting from scratch and doing this project over again, we would focus heavily on making the chatbot as user friendly as possible. We found through our testing that our bot did not sound as friendly as a support chatbot should be. This was fixed after the testing results came in but could having the idea to make it more friendly from the beginning would have saved us from making last minute changes.

If we had more time with this project, we would have spent more time perfecting each intent by fine tuning the troubleshooting steps to make them as specific as possible. Through testing we received some great feedback on what to change for certain intents however we feel a little more testing and more feedback would have been great.
We plan to keep all the coding and configurations for this project at hand for anyone to expand on should they want to. Unfortunately, we will have to remove everything from AWS as we have exceeded the free tier and are now being charged monthly for the resources used.

Conclusion

Lessons Learned

We found out very quickly that focusing on communication is key to success. Ensuring all team members are constantly checking in with each other, relaying information or just giving status updates helps drive the other team members. On the technical side we learned that creating an API is not as easy as we had hoped and that it can get very

frustrating when things do not work out the way you think. Tackling the connection between AWS and osTicket was our biggest hurdle but instead of getting stuck on it we tried to look at it from another angle. Additionally, we underestimated the power of AWS at first. After stumbling around the AWS environment trying to learn the ins and outs, we started to experiment with other services to reach our end goal. One example is using AWS CloudWatch to review the logs sent from the chatbot to verify it is in fact picking up the variables.

Skills Developed

We all got our hands dirty with the ins and outs of AWS which is a service none of us had experience with prior to this project. Learning how to use Lex, Lambda, user permissions, EC2 instances and even the cost calculator gave us the basic level knowledge needed to be comfortable stepping into an entry level position using AWS.

Also, working with JavaScript was something new for the cybersecurity students Aaron and Michael. Our Lambda connection between the chatbot and the osTicket instance relied heavily on our original JavaScript code giving us all great experience in the language.

Finally, integrating the chatbot with Facebook Messenger gave us some unique challenges not thought of until they came up. In order to get Facebooks approval to have the chatbot available to the public you must show what kind of data it collects as well as what is done with the data. This is one of the first times we all ran into an actual ethical

situation on our own and will help us greatly in the future when dealing with similar problems on a larger scale.

Future Development

Continuation on the project after graduation will be heavily focused on increasing the number of intents, updating the intents as more feedback comes in and ensuring the information the intents display to the user stays relevant. Another focus would be increasing the amount of ticketing applications our Lambda function can link up with. As mentioned earlier we are currently only using osTicket however one of the initial goals of the project was to allow this chatbot to be as flexible as possible.

Appendix: References

Mazzetti, M., & Manager, C. (2020, July 21). Home-bound customers turn to messaging channels. Retrieved September 11, 2020, from https://www.zendesk.com/blog/home-bound-customers-turn-to-messaging/

Smith, T., & Director, S. (2020, September 10). The impact of COVID-19 on customer experience. Retrieved September 11, 2020, from https://www.zendesk.com/blog/zendesks-benchmark-snapshot-impact-covid-19-cx/