University of Arkansas – CSCE Department
Capstone II – Final Report –Fall 2019

Carrier 360 Mobile Virtual Assistant

Caleb Fritz, Brandon Cox, Matthew Sij, Alicia Gillum, Anjan Poudel

Abstract

Carrier 360 Mobile Virtual Assistant

The J.B. Hunt Carrier 360 Mobile Application allows drivers to search, bid, and book loads. This application allows drivers to easily find the jobs they want at the prices they want, which greatly improves their efficiency and reduces their down time. However, due to safety concerns, the application locks down when the drivers are on the road. This prevents them from operating their devices while driving, but does necessitate that they pullover and stop while trying to find and bid on their next job. We have implemented a voice interface for the J.B. Hunt Carrier 360 Mobile Application, so that drivers will be able to leverage the functionality of the application, without having to pull over and wait until they successfully get a load. We built the application using a client-server architecture, which was done to conserve client resources and to implement a solution using Microsoft's machine learning natural language understanding service: LUIS. Our client was built using the React-Native framework and the back-end was implemented with the Azure Bot framework. Due to security concerns with using J.B. Hunt's live API endpoints, we created mock data to test and implement our application. With this voice interface, we hope to alleviate pain points of J.B. Hunt's current users and further improve the functionality and usefulness of the J.B. Hunt Carrier 360 Mobile Application.

1.0 Problem

The current process for booking new loads through the J.B. Hunt Carrier 360 Mobile Application requires that the driver manually enter key information throughout the booking and bidding process. Because of the inherent danger of interacting with mobile devices and applications, drivers must be stationary to search, select, or bid on a new job. This is a huge pain point for current and future users of the J.B. Hunt Carrier 360 Mobile Application, which aims to modernize freight services.

This stationary requirement causes drivers to stop and interact with the J.B. Hunt Carrier 360 Mobile Application. Drivers must maintain a delivery schedule, and winning bids is a time consuming and time sensitive process. This can cause drivers to either hurt their delivery schedule, or potentially miss out on lucrative bids because they are unable to frequently pullover.

2.0 Objective

The objective of this project is to save time for the drivers and add user friendliness to the Carrier 360 App. Because of easiness in controlling a device with a natural language rather than interacting with the screen, virtual assistance has become a crucial need of clients today to cope with quickly growing technology. Adding virtual assistance features in Carrier 360 Mobile mobile application will allow drivers to search for and book loads without even touching their device. It will enable drivers to safely command the app to find a load while they are behind the wheel.

3.0 Background

3.1 Key Concepts

In its current state, the application locks itself into a safety mode when the driver gets on the road. We will be using a frontend React Native client to serve as the replacement for this safety state. The React Native Framework allows us to design a cross-platform application with ease by using its seamless native code integration. We will be using Microsoft's Azure Web App Service platform to host our application. This platform will provide our application with access to the Azure Bot Service framework and Microsoft's Language Understanding Intelligent Service (LUIS). The Azure Bot Service framework will receive utterances sent by the client and will send them to LUIS to be processed. LUIS and the Azure Bot Service framework will interact with each other to properly assess the client's utterances. LUIS will translate the utterance into intentions and track down the entities they affect. These entities are components that we expect to be included in an utterance specific to the task at hand. They are parameters that we will set in the training process for our application. We will also incorporate Firebase Analytics to gather information on how users interact with the app. This information incorporated with the analytics gathered by the Azure Bot Service framework will be useful in further improving the application.

3.2 Related Work

J.B. Hunt hosts popular hackathons at the company headquarters in Lowell, Arkansas, where participating employees are given 24 hours to make their project ideas come to life. Several members of the Carrier 360 Mobile team engaged in the Spring 2019 hackathon with the idea of providing voice functionality for the app. The team was successful in their implementation, using Google's Dialogflow natural language understanding system to process user utterances, however, due to the time constraints, the full idea and functionality of their implementation could not be fully realized. This project hopes to take what they have learned and innovate on the idea while bringing new technology, like Microsoft LUIS and the Azure Bot Service into the project's scope.

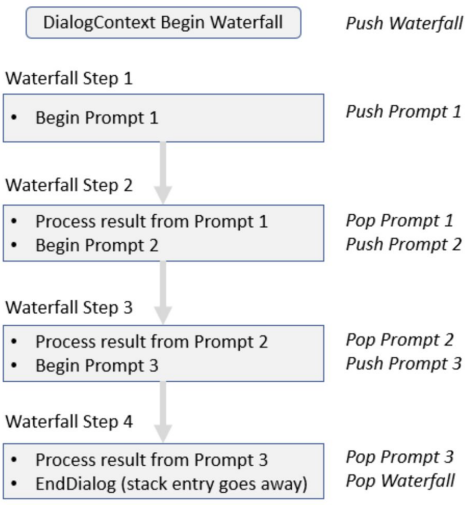
4.0 Design

4.1 Use Cases

Several use cases will be fulfilled by the development of this application, allowing a user to perform the following tasks with their voice commands:

1. Users will be able to find loads from a source location to a destination location
    a. if a source location is not specified, they must provide a required destination
2. Users will be able to specify an equipment type when searching for a load
    a. if no equipment type is specified, all equipment types will be included in the search
    b. Equipment types include:
        i. Dry Van
        ii. Flatbed
        iii. Reefer
        iv. Power Only
3. Users will be able to specify the number of empty miles they are willing to travel for a load
    a. If unspecified, a default of 100 miles will be included implicitly
4. Users may specify a load number to search rather than specifying search parameters
5. Users can decide to book the load immediately (if applicable), or view more load details
6. Upon asking for more details on a load, users may specify to hear the following details
    a. origin location
    b. origin pickup date
    c. destination location
    d. destination dropoff date
    e. price
    f. loaded miles
    g. number of stops
    h. equipment type

4.2 High Level Architecture



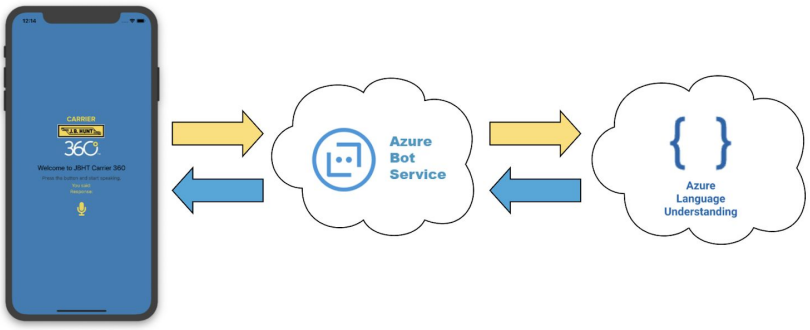
This application will allow carriers and their drivers within the integrated capacity solutions (ICS) segment to perform various actions using voice commands safely while driving their vehicles. The application will expose some of the current Carrier 360 Mobile app features to users through an intuitive voice interface that will grant them the ability to use the app with as

little physical contact as possible, allowing drivers to focus on the road ahead. Currently, the Carrier 360 Mobile app will activate a safety feature that will shutdown usability when it detects the user is in movement, forcing them to interface with the app while stationary. Our application will allow interfacing while in movement, allowing users to perform their logistical tasks within the applications in a safe and effective way, saving them time and boosting productivity.

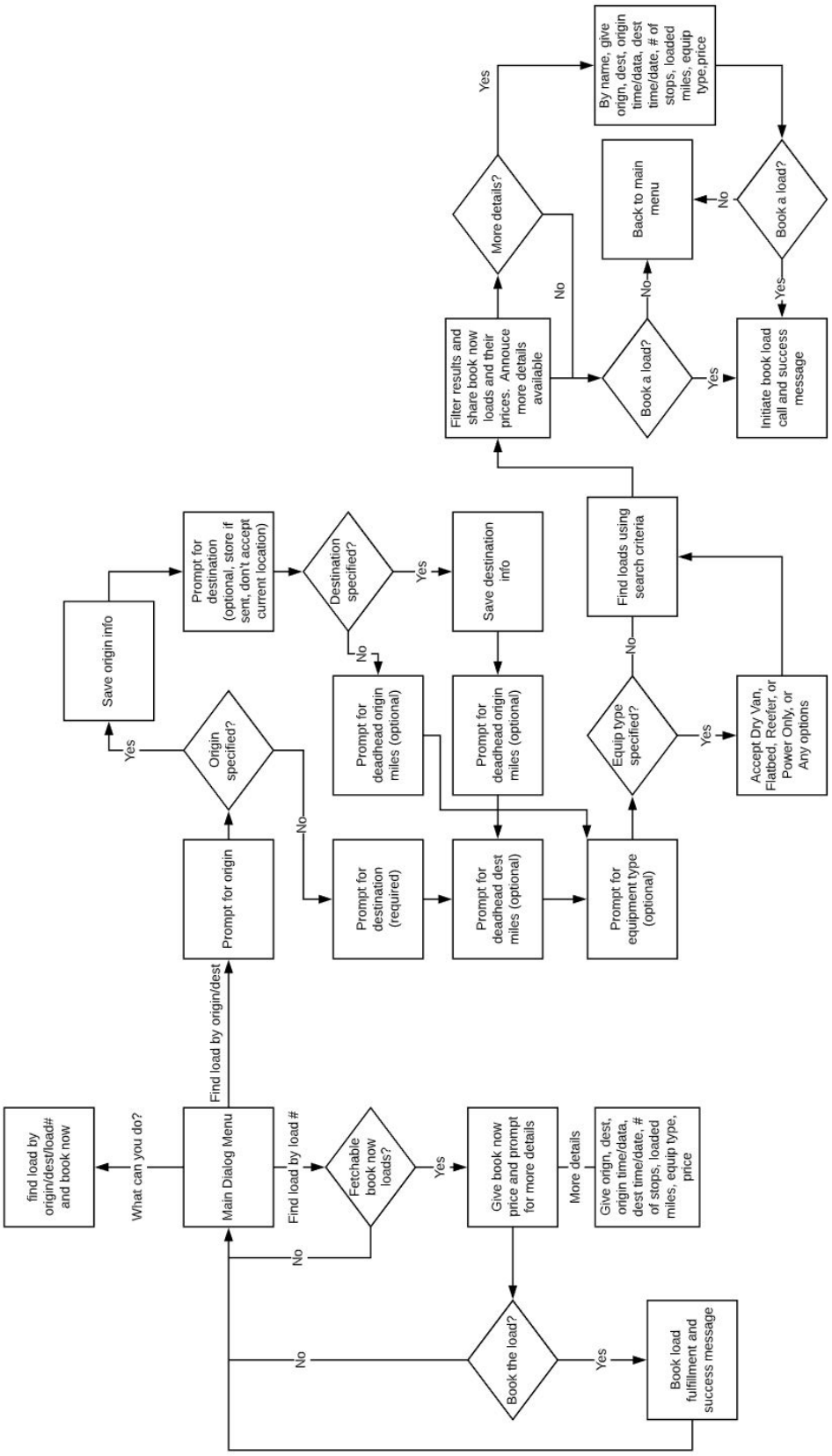
To achieve these goals, a frontend React Native application will be developed to replace the current safety screen, allowing users to interact with the Carrier 360 Mobile app while driving. This frontend piece will implement functionality that will capture user utterances and pass them to an endpoint provided by a backend Node.js server. User utterances will be captured using third-party modules either provided by the popular package manager NPM or by using voice capture cloud services. Once the backend has processed a user request and produces a response string, it will pass it back to the client, and the client will relay the response using a text to speech module.

The backend Node.js server will be hosted via Microsoft's Azure Web App Service platform and will process user utterances using the Azure Bot Service framework and Microsoft Language Understanding Intelligent Service (LUIS). The bot service will give granular control over the design of conversational flow, providing support for prompting users for input, defining question/answer sequences, modularizing conversational flow logic, and handling interruptions and fallback scenarios [1]. To understand the intentions from user utterances, the LUIS API will process the utterance using advanced natural language understanding algorithms to extract meaningful information like keywords, which will then be passed back to the bot service, providing the next logical step in the conversational flow [2]. To fulfill user requests, fixture data will be used to mimic the information that a service would normally return, which will then be passed into a load searching function to parse the information and present it to the user. Fixture data had to be used due to avoid exposing personally-identifying information like emails, phone numbers, and secure carrier access codes (SCACs), among other important information.

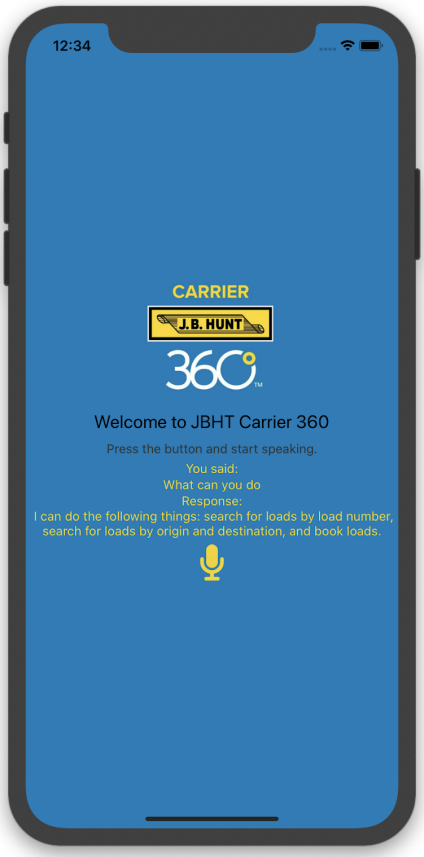
The client will make its requests to the bot server once it has been deployed to Azure via the DirectLine API. This api handles the process of identifying the bot server and keeping track of the conversation between the client and the bot. It also passes back useful information like whether or not an immediate response is necessary from the user. The bot will then send the user query to LUIS for processing, returning the intent of the utterance and any associated entities.



The flow chart on the next page shows the different voice workflows the app supports.


Main Dialog Menu
What can you do?
find load by origin/dest/load# and book now
Find load by origin/dest
Prompt for origin
Origin specified?
Yes
Save origin info
Prompt for destination (optional, store if sent, don't accept current location)
Destination specified?
Yes
Save destination info
No
Prompt for deadhead origin miles (optional)
Prompt for deadhead origin miles (optional)
No
Prompt for destination (required)
Prompt for deadhead dest miles (optional)
Prompt for equipment type (optional)
Equip type specified?
Yes
Accept Dry Van, Flatbed, Reefer, or Power Only, or Any options
No
Find loads using search criteria
Filter results and share book now loads and their prices. Annouce more details available
More details?
Yes
By name, give origin, dest, origin time/data, dest time/date, # of stops, loaded miles, equip type,price
No
Book a load?
Yes
Initiate book load call and success message
No
Back to main menu
No
Book a load?
Yes
Book load fulfilment and success message
No
Book the load?
Yes
Give book now price and prompt for more details
More details
Give origin, dest, origin time/data, dest time/date, # of stops, loaded miles, equip type, price
Fetchable book now loads?
Yes
No
Find load by load #

4.3 Client Design and Implementation



The client user interface is designed for a user experience that is intuitive. We chose to use React-Native to make the application more versatile between devices. The user should only have to interact with the device via voice, so the interface should be simple. We chose to design the interface using the color scheme of the existing app, using the blue, yellow, and white throughout the main voice screen. The microphone button, when clicked, activates a listening function that will capture user utterances. The JBHT logo sits near the top of the screen in order to indicate the app's association with the desktop version of the carrier application. The screen is implemented using React-Native components and the interactive buttons use TouchableHighlight objects. Lottie animations were used to animate the button. Learning to program with React-Native and using Lottie animations were new experiences for some members of the team, so these skills will be quite helpful for further expanding our knowledge of mobile programming. Ideally, this app could be improved with more professional fonts and animations; however, we have achieved an intuitive user experience which is one of the crucial aspects of user interface design. This is but a small, but important stepping block for what could become of the JB. Hunt Mobile Carrier 360 application.

4.4 Server Design and Implementation

The backend server was implemented using Microsoft's Azure Bot Framework. The Bot Framework allows developers to create bots that are able to carry on scripted conversations with users. The framework includes tools for creating various prompts such as number prompts, confirmation prompts, choice prompts, and text prompts. Within the bot, developers can create "dialogs". A dialog is essentially a piece of the conversation. Dialogs may be nested as well. Each dialog typically contains some master conversation flow. For example, dialogs based on a waterfall dialog will go sequentially through a series of other dialogs.

The main application dialog is the dialog which users start in. The main dialog is based on the waterfall dialog model. Once in the dialog, users have the option to either search for a load to book by load number, search for a load by other parameters, or ask what the bot is able to do. Users have their queries interpreted by the LUIS service, which contains a model that attempts to classify user utterances into one of the three options. Then, the bot will either redirect the user to the appropriate subdialog, or it will read tell the user what he/she is able to do.

If the user asks to search for routes by load number, they are redirected to the “Route dialog”. Like the main dialog, the route dialog is based on the waterfall dialog model. The first step verifies that the internal list of routes is not empty. If there are no routes available, then the bot will notify the user and exit the dialog. If there are routes available, then the dialog will continue by asking the user what load number they are looking for. The bot then confirms that it heard the route number correctly and continues back to the main dialog.

If the user asks to search for routes by other parameters, then they are redirected to the “search dialog”. This dialog also follows the waterfall model. The first steps prompt users for their preferred origin and destinations. They are able to not specify an origin or destination. If they do not specify an origin, then they must specify a destination. The bot then asks the user how many deadhead miles they would like to travel at both the origin and destination. Deadhead miles are how far the driver would need to travel from the given location to the load pickup or dropoff location without cargo. The user is then asked to specify what type of trailer they have to carry the load. Then, the user is readback all of the entered information so that they can confirm that it is correct. If it is incorrect, the dialog is restarted. If it is correct, the user is returned back to the main dialog.

Once the user has entered the parameters necessary to search for a load by number or by parameters, the main dialog queries the list of available routes for all routes matching the given parameters. The price and load number for the results are shown to the user. If more than one route is returned, the user is asked to select one route. The user then has the ability to ask for more information, book the route, or exit the dialog. If the user asks for more information, the user is given all of the known parameters about the route and is then asked if they want to book the route now. If the user asks the book the route, the bot books the route. Otherwise, the bot exits the dialog.

The Azure Bot Service contains an API that allows external devices to message and maintain conversations with published bots. The API, known as the DirectLine API, initializes conversations with authenticated devices by first giving users a conversation token. Users must pass this token back to the server every time they send a message. Each message sent to the server is passed to the bot as a message in the conversation. The server responds to each message with whatever its response is to the message sent by the client.

4.5 Lessons Learned

Since J.B. Hunt test endpoints closely resemble production endpoints, much of the test data is populated with personally-identifying information like emails, names, and phone numbers. Some internal information like DOT numbers and secure carrier access codes (SCACs) are also visible. To mitigate this issue, the original plan to implement the actual endpoint services was replaced with creating fixture data used to represent the data that would be returned and extracted from the service call. This allows the bot server to still function as if it were making actual endpoint calls without exposing sensitive information. Unfortunately, this does reduce the authenticity of the app, but the system was designed such that it would be painless to extend the code to implement the services and the data parser necessary to allow the bot server to make actual service calls.

In the proposal, we also mentioned that we would be integrating analytics. However, the addition of analytics proved to be beyond the scope of the project due to time constraints imposed by other aspects of the project. In a further iteration of the project, we would hope to get a clear understanding from a business perspective on what aspects of the app should be tracked by analytics so that they may then be implemented within the proper workflows.

4.6 Potential Impact

Though the application is not production ready due to the endpoint restrictions discussed, it can serve as a basis for future voice applications, and can be compared with existing voice technologies using competing frameworks like Google's DialogFlow to determine which approach would be most effective. The bot server's design effectively separates its implementation from the client implementation, allowing it to be maintained separately without impacting users.

4.7 Future Work

Several important improvements can be made to enhance the functionality of the bot server. First and foremost, implementing services to call actual test and production endpoints will be necessary to make the voice bot useful and profitable. Next, to gain a better understanding of how the bot is used, analytics should be implemented using Azure Application Insights to discover the most and least used features for further improvements. Though LUIS allows users to provide a wide array of responses, the bot server's dialog workflows are still somewhat rigid in that a user must complete a workflow once they have started one, and cannot go back to previous steps. The dialogs could be enhanced by allowing backtracking and providing better handling for when a user wishes to escape a workflow. Finally, the test-to-speech (TTS) package employs a generic voice that is somewhat monotone and less authentic than perhaps other solutions and serves as a point of improvement as well.

4.3 Risks

Risk	Risk Reduction
Restricted Access to J.B. Hunt endpoints disallows full testing of voice app for most team members.	Use Tundra-Fetch endpoint capture software to generate data profile fixtures with PII attributes removed/obfuscated to mock services.
Sensitive API keys and endpoints will be used within the client and server to perform operations, posing a potential security risk.	Employ best security practices by keeping sensitive resources within git-ignored environment files and use them securely in a test environment. For production, store keys remotely and employ encryption.
App usage while driving could pose legal questions.	Design the application within the scope of State and Federal laws and follow requirements imposed by J.B. Hunt for optimal safety.

4.4 Tasks – The following tasks are to be completed for this project:

1. Gain background knowledge of the required frameworks and APIs for the project and setup project environment
    a. Understand React-Native framework and the following libraries/middleware
        i. React-Redux for application state management
        ii. Redux-Thunk (potentially Redux-Saga) for async action handling
        iii. react-native-voice for converting user speech to text
        iv. react-native-tts for converting text to speech for relaying messages from the server to the user
        v. ESLint for code styling rules
        vi. NativeBase for basic, customizable components
        vii. node-fetch for http requests
    b. Understand LUIS to build/train a language agent to extract useful text entities from identified intents produced from user utterances
    c. Understand Azure Bot Framework SDK to design conversation flow modules
    d. Understand Firebase Analytics SDK to integrate analytics into project
    e. Create a project repository using Azure DevOps and establish code rules and PR policies
2. Design client app
    a. Initialize a React Native project and install necessary libraries, establish organizational and code styling guidelines via ESLint and a clear readme for easier collaboration
    b. Design app interface by developing mocks and use case scenarios
    c. Determine the state tree for the app
3. Implement client app design
    a. Develop screen components and style using StyleSheet attributes and components from NativeBase library
    b. Integrate react-native-voice and react-native-tts into project to capture voice data, convert to text, then back to voice data.
    c. Integrate React-Redux into app to manage state
        i. Develop synchronous and asynchronous action creators to dispatch actions
        ii. Develop reducers to capture actions and update app state
    d. Use DirectLine REST API and node-fetch to send user utterances to the Node.js bot server, and receive text responses from the server
4. Design bot server
    a. Initialize a Node.js restify server and integrate Bot Framework Service middleware
    b. Implement prompts and dialog workflows using Bot Framework SDK dialog library
    c. Connect LUIS to the bot service to extract user intents and entities for usage in the dialog workflows
    d. Implement search functionality that will import provided fixtures to search against the user queries

    e. Capture response data from fulfillment and package up into a text response to be sent back to the client
5. Design and train LUIS voice agent
    a. Create LUIS app and add app credentials to bot server
    b. Create intents to identify possible user utterances
        i. Develop fallback intents for unknown or irrelevant utterances
    c. Apply intent entities to identify key information within utterances
6. Deployment, testing, and documentation
    a. Deploy bot server to azure
    b. Test and debug bot server individually using Bot Framework Emulator
    c. Test and debug client app individually using react-native-debugger
    d. Test all possible dialogue workflows with client app and bot server integrated.
        i. Test "happy-path" scenarios, where dialog workflow criteria is successfully met
        ii. Test "sad-path" scenarios, where dialog workflow criteria is not met at various stages, and fallback scenarios successfully handle these situations
    e. Test analytics is detecting app usage and displays meaningful telemetry
    f. Generate documentation for final project

4.5 Schedule – Below is an overall schedule of the tasks to be completed. Notice some tasks will be performed in parallel with others. Completion dates are subject to change depending on progress variations during implementation.

Tasks	Assigned to	Dates
1. Gain background knowledge in frameworks and SDKs	Everyone	8/26 - 9/16
2. Design client app	Alicia, Anjan, Matthew	9/17 - 10/1
3. Implement client app	Alicia, Anjan, Matthew	10/2 - 10/30
4. Design bot server	Caleb, Brandon	9/17 - 11/14
5. Design and train LUIS agent	Caleb, Brandon	9/17 - 11/14
6. Deploy, test, and document	Everyone	11/15 - 12/13

4.6 Deliverables – The following components will be provided at the completion of this project:

- Website code: The PHP code for the team capstone website
- Non-proprietary JavaScript code for client application

- Non-proprietary JavaScript code for bot service server
- Non-proprietary intent and entity data for LUIS agent
- Final Report

5.0 Key Personnel

Caleb Fritz (Team Lead) - Fritz is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He is engaging in undergraduate research pertaining to Computer Vision driven mobile app automation testing. He has completed relevant courses including Software Engineering, Algorithms, Information Security, and Operating Systems. He is currently an information systems intern at the J.B. Hunt headquarters in Lowell, Arkansas working as a frontend developer and automated test scripter. He will be working on the frontend client app and the backend bot service server, with a specific responsibility for service fulfillment using J.B. Hunt endpoints.

Matthew Sij - Sij is a junior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He is minoring in both Mathematics and Physics. He has completed the following relevant courses: Software Engineering, Information Security, Algorithms, and Artificial Intelligence. He currently works as a research assistant in the department of Computer Science and Computer Engineering, where he works on Computer Vision and Deep Learning related fields. Sij will be interning at Metova, Inc. as a developer intern this summer. He will be responsible for the client side of the project.

Alicia Gillum – Gillum is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. She is minoring in both Mathematics and Physics. She has completed the following relevant courses: Software Engineering, Database Management Systems, Wireless Systems Security, Information Security, and Algorithms. She currently works at the University's Security Operations Center as a security technician. She also works as an undergraduate research assistant in the department of Computer Science and Computer Engineering, where she works on projects related to Cybersecurity. Gillum will be interning with Tyson Foods on the security team during the summer. She will be working on the client side of the project.

Brandon Cox – Cox is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He is minoring in Mathematics. He has completed the following relevant courses: Software Engineering, Operating Systems, Algorithms, Information Security, and Artificial Intelligence. He has worked at Datapath, Inc. as a Developer Intern working as a full-stack developer using ASP.NET. He is currently working at Metova, Inc. as a developer intern as an API developer using ASP.NET. Cox will be responsible for the backend side of the project.

Anjan Poudel - Anjan is a Senior Computer Science student in the Computer Science and Computer Engineering Department at the University of Arkansas. He works as an Application Developer at the University of Arkansas IT Services. He has completed following classes Computer Vision, Artificial Intelligence, Design and Analysis of Algorithms, Operating Systems, Database Management Systems, Software Engineering, Information Security. He will be working on the client side of the project.

6.0 Facilities and Equipment

J.B. Hunt endpoints for the Carrier 360 Mobile application will be used to provide the fulfillment pieces from user utterances.

7.0 References

[1] Taylor, J., Iqbal, K., & Berry, I. (n.d.). Dialogs within the Bot Framework SDK - Bot Service. Retrieved from https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-concept-dialog?view=azure-bot-service-4.0

[2] Diberry. (n.d.). What is Language Understanding (LUIS) - Azure Cognitive Services. Retrieved from https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis