Feasibility Study

Edit

Jonathan Engelsma edited this page just now · 2 revisions

Team Motorola

Katelyn M., Jesse R., Adam T.

Overview

Team Motorola looks to build an assistive cognitive-bot for 911 call-takers. Through leveraging Amazon Web Services (AWS) technologies and frameworks, we can interact with multiple callers simultaneously though voice and text. By utilizing a modified MEAN JavaScript web stack, we will develop an intuitive web application that can provide 911 call-takers with meaningful insights. In doing so, call-takers can better assist 911 callers by providing better and more informative service, thus improving and even saving lives.

Languages

The main bot engine will be built using an inherent Amazon rule-based language built within Amazon Lex. Text analysis and interfacing between Lex, databases, APIs, and the web application will be developed using Python scripts within Amazon Lambda containers.

The project's web application will be constructed with a JavaScript stack. Node.js, Express.js and Angular.js 1 - all JavaScript libraries - will be used to build the application server, REST API routing, and view templates. Other JavaScript libraries such as Underscore.js and JQuery may be utilized to process data and handle simple front-end DOM manipulation such as toggling menus or hiding elements on the page. HTML and CSS will be used to implement each application view prototype.

Frameworks, Libraries, and APIs

As requested by our Motorola client, the entire process flow and framework will be entirely hosted on the AWS platform. Amazon Lex contains a voice to text interface, as well as a rule-based chat-bot application. Amazon Lambda will be utilized as containers for Python and Node.js code to interact with other Amazon services and the web application. As mentioned in the previous section, a number of JavaScript libraries will be used to construct this project's web application. The Node.js library will be used to construct the application server and process data received from Lex. Express.js - a JavaScript routing library - will be utilized to create a REST API that will facilitate communication between the application front-end, the Node server and AWS Lambda containers. The front-end of the web application will be templated and routed with Angular. Each application view will utilize Angular's two-way data binding to inject data dynamically into HTML templates. Angular will also handle high-level processing and filtering of data displayed to the user. Finally, the

Pages 10

Home

- Cognitive 911 Bot Wi

Final Deliverables

- Project Video (Youtu Format)
- Project Video (HD)
- Final Report
- Final Presentation

Deliverables

- Prospectus
- Feasibility Study
- Sprint 1 Report
- Sprint 1 Presentation
- Sprint 2 Report
- Sprint 2 Presentatior
- Sprint 3 Report
- Sprint 3 Presentatior

Journals

- Adam's Journal
- Jesse's Journal
- Katie's Journal

Miscellaneous

- Group Meetings Note
- Poster

Clone this wiki locally

https://github.com/jer

Clone in Desktop

Bootstrap HTML/CSS library will be utilized to build a responsive and mobile friendly application views.

Code Repository Organization

Our main code repository will be hosted on a private GitHub repository. Within the repository will be the Node server code and the front-end Angular code. Each of these directories will contain sub directories for each component of the application - See Figure 1 for details. Although the Amazon Lex bots will be hosted on Amazon servers, we will use GitHub to maintain the Amazon Lambda Python text analysis and Node.js interfacing code. We will be using an Amazon EC2 server to host the web application and Amazon DynamoDB to host call transcripts and raw data.

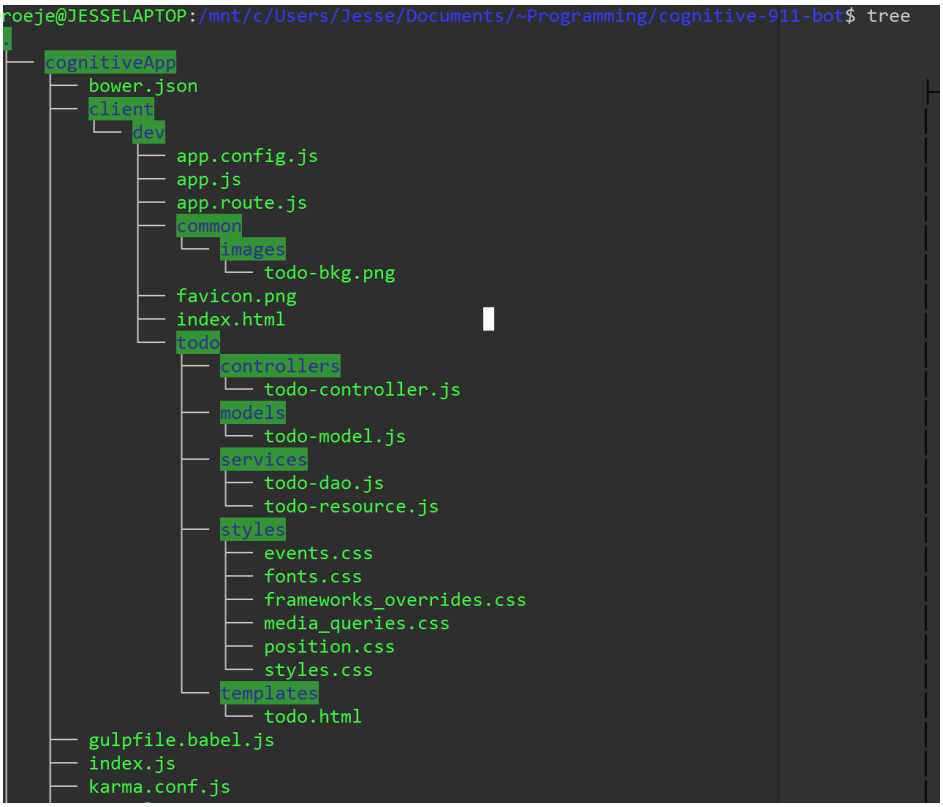

Figure 1: Sample Repository Structure

Preliminary Database Structure

One of the tables in our schema will be a table used to simulate a caller queue. The second table in our schema will be a mapping from the call to the operator that picked up the call. The last two tables in our schema will hold all of the information collected about a call. One table will hold data that will be associated with every call and the second will hold a mapping of the call id to important information about the call.

```
CallerQueue(CallId, PriorityRating)
CallToOperator(CallId, OperatorId)
CallStaticInformation(CallId, PhoneNum, EmergencyType, EmergencyLocation, Timestamp)
CallInformationMapping(CallId, InformationType, InformationDetails)
```

System Organization

Multiple callers first interact with a chat-bot through SMS for a period of time. It may be as short as a single text message saying they are now being connected with a call-taker, or it may be multiple text messages sent back and forth. The SMS interface will, at first, be simulated through a web application, but will be eventually deployed on mobile applications or through text messaging. Additionally, we look to consider a voice-to-text chat-bot interface where the caller may be receiving texts from the chat-bot and communicating via call or the chat-bot might use text-to-voice functionality to communicate with callers. Python scripts contained in Amazon Lambda will process the text transcripts collected from Amazon Lex and send meaningful keywords and key information to a REST API deployed using Amazon API Gateway. This REST API will communicate between Amazon Lambda functions and the Node server. The Angular front-end will use a REST API to GET or POST meaningful data (keywords, location, injuries, etc.) to the Node server. This data will be displayed in a handful of views as seen in the wireframes that will be discussed later. The entire system organization can be visualized in the following diagram (Figure 2).

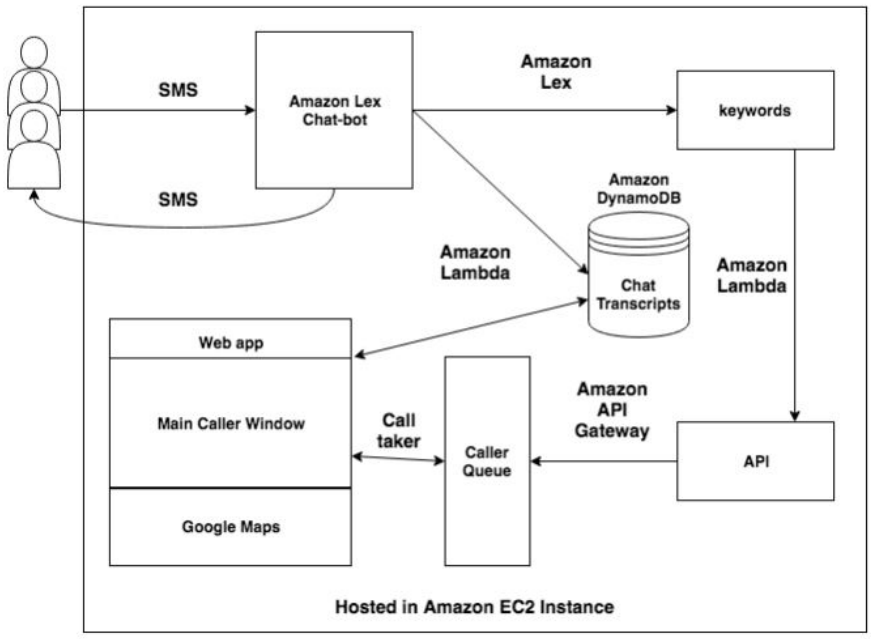

Figure 2: System Overview

Development

The web application portion of the project will be developed locally on our computers in a development environment. The web application will be hosted on an Amazon EC2 instance for both development and production. Amazon Lex serves as both a prototyping and deployment environment for bots. Chat-bots will be tested using a sample web interface and will eventually be deployed as a mobile SMS interface. Amazon Lambda Python and Node.js scripts will be built in development and production branches on git and will be constantly and consistently tested using AWS.

Non-Trivial Requirements

Chat-bot

Amazon Lex will be used to process voice to text data. Additionally, Amazon Lex is built to function as a chat-bot through "sample utterances", "slot type values", and "error handling". Essentially a set of rules will be used to communicate with the caller through SMS messaging (simulated as a web interface). Tutorials have been completed and sample chat-bots have been implemented to test the functionality of Amazon Lex.

Text analysis

Amazon Lambda Python scripts will be used to identify key information about the emergency such as the location, the number of people who were injured, and the type of emergency, among other things. These key pieces of information will be displayed in the call details view of the web page. Example Amazon Lambda functions have been developed and implemented in both command-line and Amazon GUI approach. These Amazon Lambda functions offer significant freedom, as they serve as containers for code that interacts with any of AWS's services. As such, further exploration with Amazon Lambda is needed to truly understand its capabilities.

Web Application

One of this project's requirements is to be able to display and receive data from a user. A web application will be developed to fulfill this requirement. The web application can be broken into two main elements, back-end server and front-end web client as seen in Figure 3.

The web server will be developed in Node.js and will provide data processing and host routing logic for both REST APIs and application routing. The server will be composed of a number of controllers that will each define REST API routes for various elements of the application such as communicating with Lex, requesting data from database and loading the Angular app. Each controller will define functions for processing received data and will utilize the Express.js routing framework to define API endpoints, handle application responses, and construct middleware functions.

The front-end component of the web application will be developed in Angular.js 1. Angular will allow data to be injected dynamically into HTML templates and filtered based predefined rules. The front-end application will be designed as a single page site. Each page will extend a main view that will contain navigation elements. Angular routing functions will handle navigation between application views and will trigger calls to the Express REST API to fetch or post data. Once data is obtained from the API, it will be passed through filtering functions and displayed to the user using Angular's two-way data binding. This will allow for a quick application that does not require page refreshes as data is updated. This is especially important, as the data displayed to the user will be updated regularly as it is produced by the Lex chat-bot.

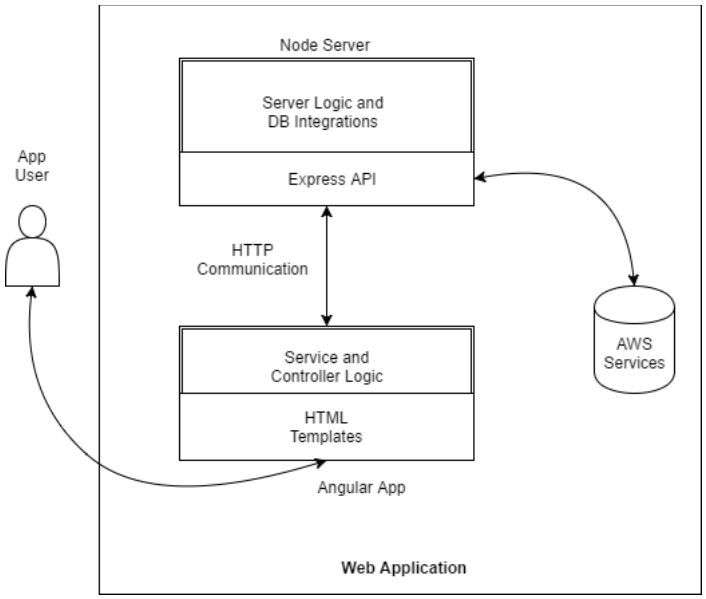

Figure 3: Web Application Design

Wireframes

Caller Queue

When people call 911, often there are not enough operators to talk to callers immediately. While the callers are waiting to speak to an operator, this time can be used to gather preliminary information about the emergency and place the callers into a priority queue based on the type of emergency. Operators will be able to view this queue and see a summary of the information collected. Seen below is a preliminary wireframe of the caller queue.

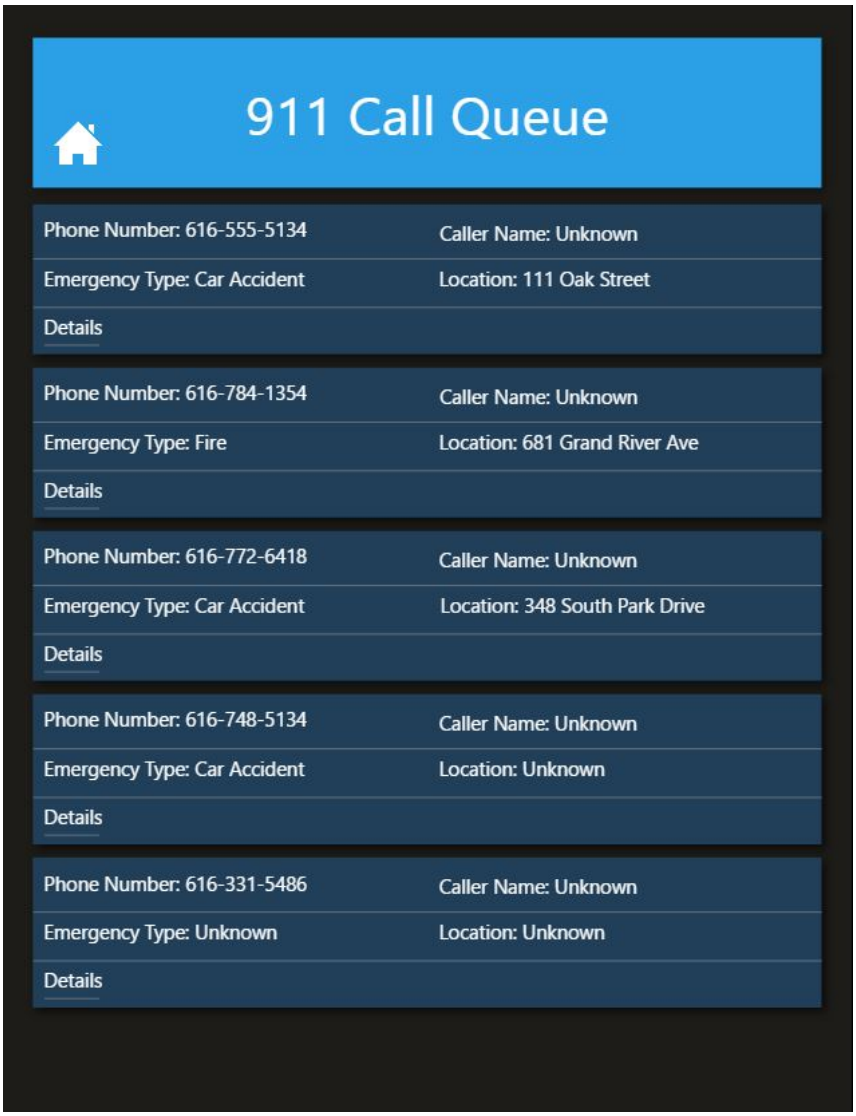

Figure 4: Call Queue Wireframe

Call Details

When an operator wishes to answer a 911 call or view the details of a call, they can pull up the call details view. This view will contain a summary of all of the information collected by the chat-bot. As the call progresses, more details will be filled in by the model.

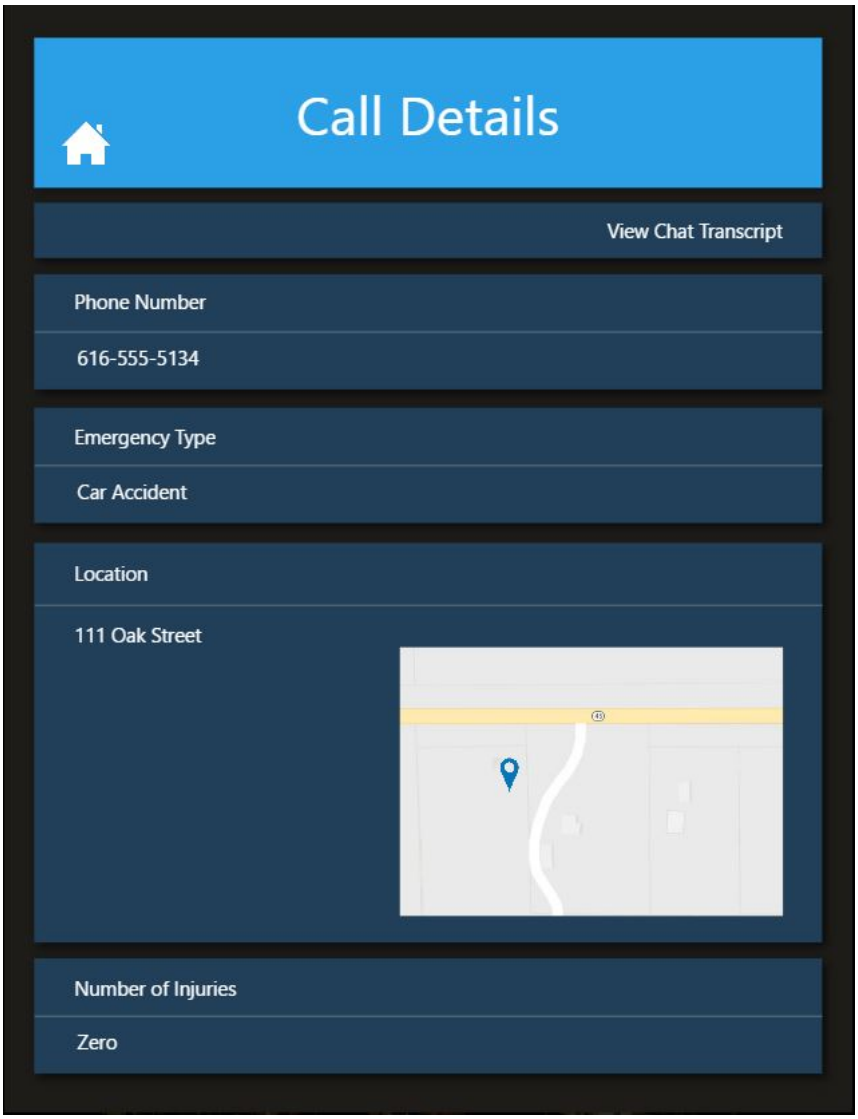

Figure 5: Call Details Wireframe

Chat Transcript

In case an operator wishes to view the chat transcript or possibly the audio-to-text translation of the call, the operator can open the transcript view of the web application.

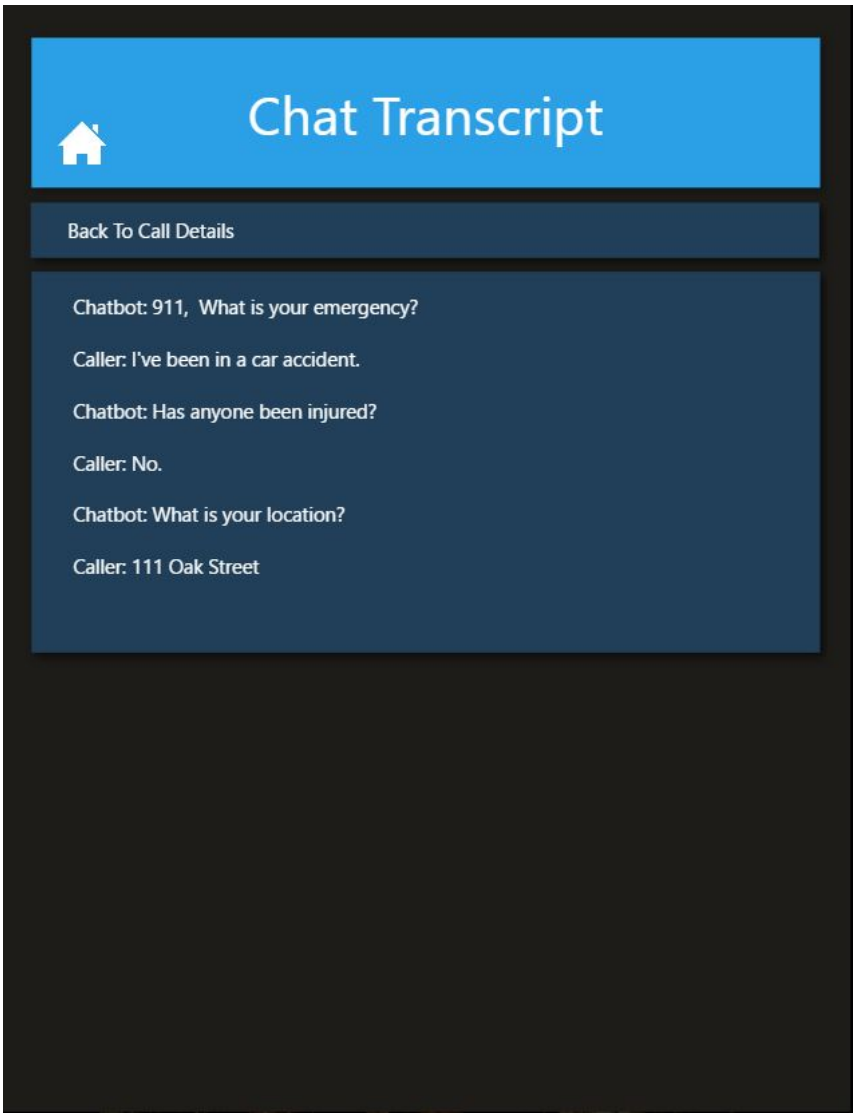

Figure 6: Chatbot Transcript Wireframe

Chat Screen

The final view of the web application is the chat screen in which 911 callers will be able to talk to our chat-bot. The chat-bot will use the time that the caller is waiting for a 911 operator to become available to collect basic information about the emergency such as the location, number of injuries or the callers name.

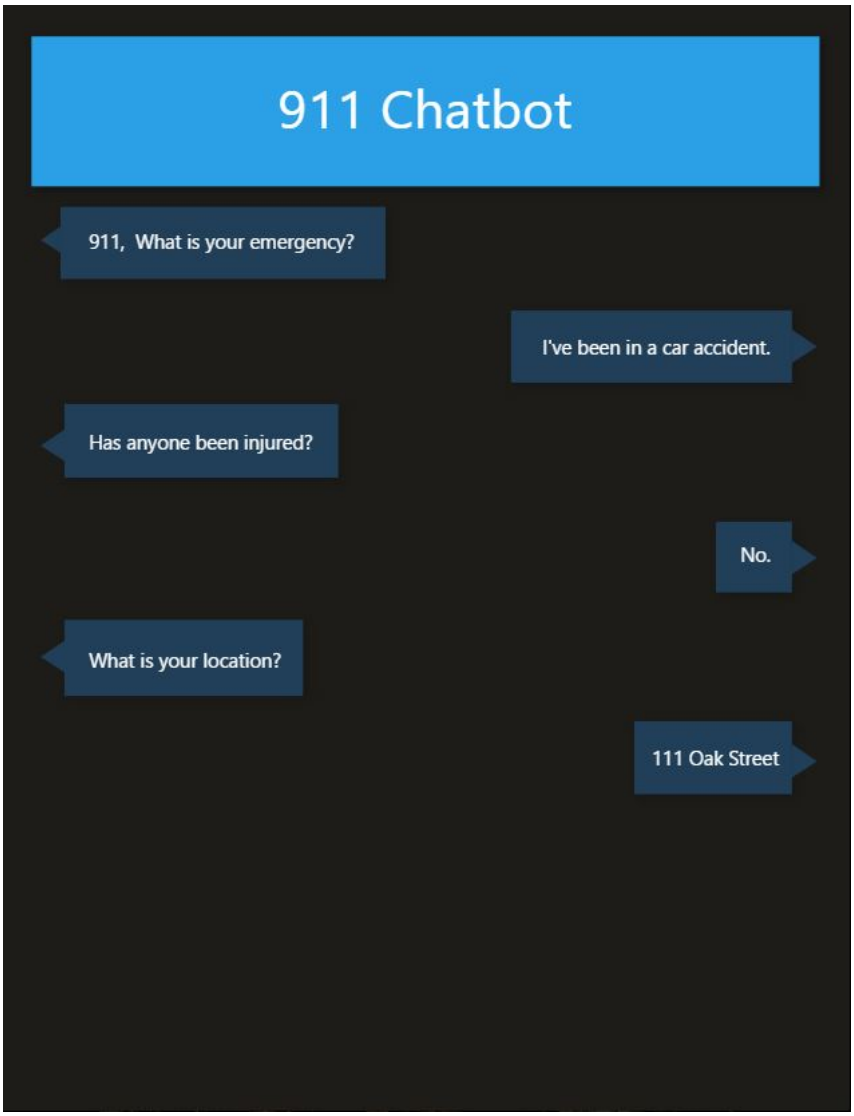

Figure 7: Chat-bot Chat Screen Wireframe

Division of Labor

The simplest division of labor is model (Adam), view (Katie), presenter (Jesse). However, this is an oversimplification, as each of our tasks overlays one another to some extent. More specifically, given Adam's skill-set and background in Natural Language Processing (NLP) and Python scripting, he is well suited to build the NLP bot-engine, as well as, text analysis Lambda Python scripts. Jesse has an interest in back-end web development and experience building MEAN JavaScript stacks, so he is well suited to build Amazon Lambda Node.js scripts and the Express routing of the REST API to communicate with the web server. Katie has valuable expertise in front-end and user interface design. As such, she will concentrate on building the main views of the web application using an Angular front-end. Katie and Jesse will work collaboratively on Amazon DynamoDB setup and communication with the web application. Since Jesse serves as the "presenter", he will integrate with both Adam's Lambda scripts and Katie's Angular front-end. Jesse and Katie will also assist Adam where needed with the NLP bot-engine and Python scripting.

Work Policies

As mentioned in previous sections, we will be using development and production branches in git for web application. Development branches will be utilized for local development and the master branch will be used to merge in new features as they are created. The production branch will represent the most recent fully functioning version of the application and will be used to host the web application code on an AWS EC2 server. Additionally, Amazon Lambda code will be developed both in local git repositories, as well as, on Amazon servers. The interaction between Lex and Lambda is self-contained on Amazon. More significant testing for the integration between Lambda and the API is required.

Test Plan

Unit and System testing will be considered for both the web application and the Lambda functions. Web application testing will be accomplished using the Jasmine testing framework. Both unit and system tests will be created to verify the functionality of both server and front-end features.

Deliverables and Milestones

Sprint 1

Creation of a basic chat-bot with 911 specific rules using Amazon Lex.

Amazon Lambda Python scripts that do basic text analysis on 911 transcripts.

Research into the creation of Lex endpoint API with AWS API Gateway framework.

The setup and configuration of DynamoDB, and evaluation of Lex data to determine schema(s) that will be required for data storage.

Development of basic Node server and Express API for interaction with AWS gateway and Angular application.

Creation of initial designs of HTML templates will be implemented and integrated into the Angular application. Basic test data will be used to populate each view for design testing.

Establish basic Angular.js application and integrate with initial HTML templates.

Sprint 2

Our team expects to compile a more extensive list of Sprint 2 components as we progress through Sprint 1.

Functional 911 chat-bot serving multiple simultaneous clients using Amazon Lex.

Basic audio to text functionality using Amazon Lex.

Amazon Lambda Python scripts that collect all meaningful data (keywords, location, injuries, etc.).

Integrate live Lex data into web application. Create processing logic within express middleware to clean data and format for use by the front-end application.

Creation of AWS API Gateway for transfer of data from Lex to web application.

Finalize database schema(s) and create integration code within Node server to handle storage and retrieval of data.

Update Angular application to handle and filter live data.

Update REST API to utilize middleware and expand to support live data (add specific endpoints for each distinct set of data).

Establish test suite to validate functionality of Node server, Angular application, and Lambda code.

Sprint 3:

Our team expects to compile a more extensive list of Sprint 3 components as we progress through Sprints 1 and 2.

911 Amazon Lex chat-bot integrated with SMS text.

Audio to text on mobile integrated with chat-bot.

Finalize design and functionality of Angular application and Node server.>

Finalize REST API on AWS and within Node server.

Expand test suites to verify functionality of final application features.

+ Add a custom footer