Iris - Mobile Application for the Blind

by

William Anderson & Tristan Guckenberger

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

William Anderson	**03/15/2019**
William Anderson - Project Manager, Software Developer	Date
Tristan Guckenberger	**03/15/2019**
Tristan Guckenberger - Business Analyst, Software Developer	Date
Abdou Fall	**03/15/2019**
Abdou Fall - Faculty Advisor	Date

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

April 2019

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

List of Illustrations

TABLES

FIGURES

ACRONYMS AND ABBREVIATIONS

API - Application Programming Interface

JS - JavaScript

ML - Machine Learning

AI - Artificial Intelligence

IDE - Integrated Development Environment

OS - Operating System

ABSTRACT

According to the International Agency for the Prevention of Blindness, an estimated 36 million people are blind and an additional 217 million have moderate to severe visual impairment. This means that a total of 253 million individuals must rely on their four available senses, primarily touch, in order to perceive their surroundings. One major limitation of this is that the individual must be within arm's reach (or cane's reach) in order to determine what is in front of them. Iris changes this by providing a mobile application with a voice-controlled user interface to improve environmental awareness for the blind. With Iris, users are able to take a picture and have the application describe it to them with additional information on specific entities within the picture if requested. Through accessibility features and the power of machine learning, Iris is an extension to the user's senses.

PROBLEM STATEMENT

Introduction

There are an estimated total of 253 million people that must rely on their four available senses, primarily touch, in order to perceive their surroundings. As non-blind individuals, there are many aspects of our daily life that we take for granted. Reading signs, counting currency, and simply knowing what's in front of us are all luxuries that the blind or visually impaired cannot experience the way we do. There are braille signs and safety measures that blind individuals can feel with their canes, but these accessibility features often aren't implemented and even when they are, the person must be only a few feet away to utilize them.

Project Description

Iris is a mobile application with a voice-controlled user interface that utilizes machine learning libraries to improve environmental awareness for the blind. Users will be able to take a picture and have the application describe it to them with additional information on specific entities within the picture if requested. Iris will also detect if there is any text in the image, in which case it will read the text to the user.

Problem

Blind individuals rely on their four available senses, primarily touch, in order to perceive their surroundings. One major limitation of this is that the individual must be within arm's reach (or cane's reach) in order to determine what is in front of them. Additionally, important warnings or notices posted on signs often don't have braille transcriptions and even if they do, a blind person must feel the sign to know that it is there. Even if the individual is within reach of their surroundings, it can be hard or even impossible to perceive things like written text or currency values.

Solution

Iris will describe a user's surroundings by processing a photo taken through their cell phone camera. A user will be able to navigate and command the application with voice controls.

Iris will deliver audio transcriptions of contextual information from a photo captured by the user. Additional components include detailed descriptions of specific entities from the photo upon request, optical character recognition to read visible text to the user, currency recognition to read the value of bills and coins, and a feature allowing the user to save a description/photo to the user's library.

Users should be able to provide feedback on the provided description which could then be used to further train the machine learning model. Through this loop, the model will continue to get more accurate as more users utilize the service.

This project will feature integration with Google's Cloud Vision API, as well as React Native Voice, a speech to text library for React Native. Additionally, other existing machine learning models and technologies may be utilized as they are already the most accurate models available.

By utilizing pre-existing machine learning models and react native libraries used by some of the biggest tech companies in the world, we are ensuring that Iris provides

best-in-class image recognition and speech recognition.

User Profile

Iris is targeted towards individuals who are either blind or have a visual impairment. The user should be experienced with smartphone accessibility options such as voice-over as well as more general smartphone concepts like Siri, Google Assistant, and touch screen interaction. The Iris software itself does not have a visual interface, but some degree of experience with visual interfaces will be likely be needed to install and open the application.

Project Title

Iris - Mobile Application for the Blind

Potential Users

Blind and visually impaired individuals

Software and Interface Experience

The target audience for Iris should be experienced with smartphone accessibility options such as voice-over as well as more general smartphone concepts like Siri, Google Assistant, and touch screen interaction. The Iris software itself does not have a visual interface, but some degree of experience with visual interfaces will be likely be needed to install and open the application.

Experience with Similar Applications

- Microsoft's Seeing AI
- Be My Eyes
- Siri (voice-based assistant)
- Google Assistant (voice-based assistant)

Task Experience

Users should be experienced with basic voice-over navigation and touchscreen interaction. Potential users may have experience giving vocal commands to an AI assistant such as Siri or Google Assistant.

Frequency of Use

Iris is intended for use in any given situation where blind or visually impaired users need further situational-context. Use could range from throughout a users day to only a few instances a month. The user's level of visual impairment could also impact their frequency of use; a fully blind user will likely use the app more frequently than a user who simply needs help reading distance signage.

Key Interface Design Requirements That the Profile Suggests

- Minimal to no visual user interface
- Single touch interaction for auditory feedback
- Optional voice commands for Iris
- Strong ease of use and intuitive design

Use Case Diagram

Figure 1: Use Case Diagram presents the Iris use case diagram. In the diagram, there is a single targeted user, someone with a form of visual impairment. Within the application, users may: Ask for a general description of their environment (and request additional information), capture their surrounding without any specific requests (and request additional information), request a text transcription, or ask for a description / quantity of currency.

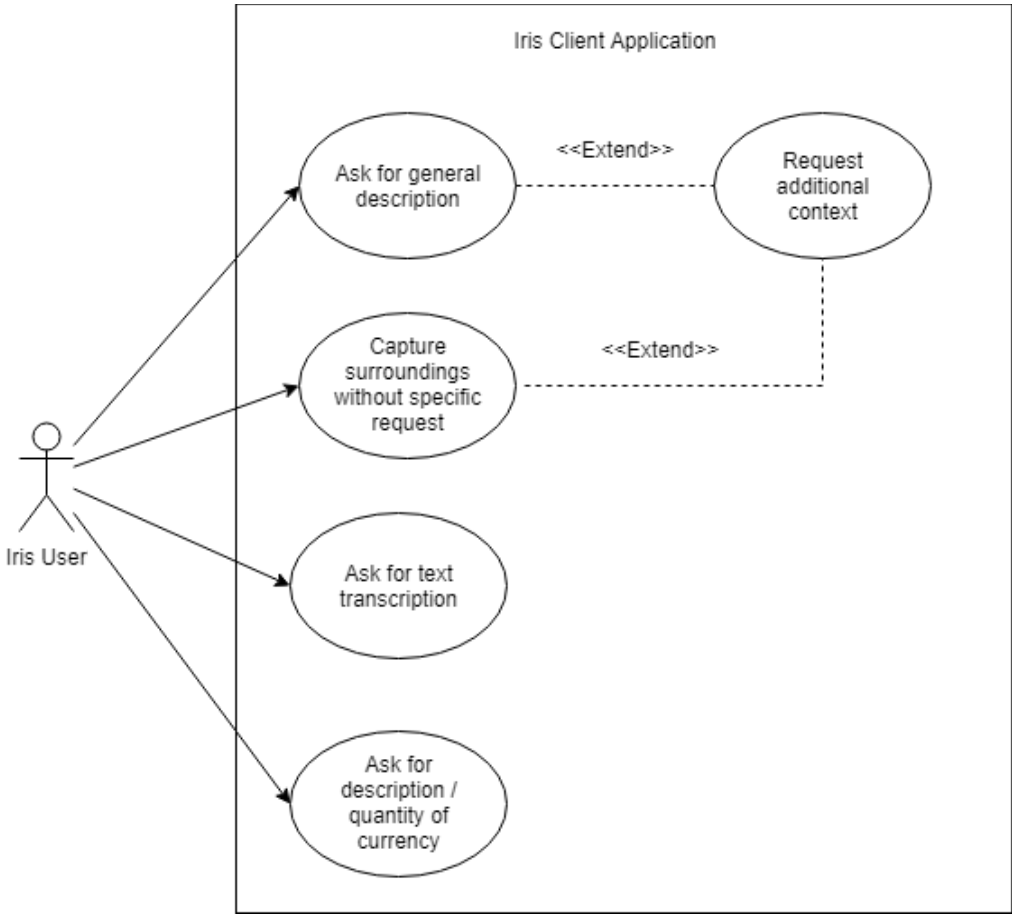

Figure 1: Use Case Diagram

PROJECT MANAGEMENT

Budget

Table 2: Project Budget presents the Iris budget. The budget shows costs from the beginning of November 2018 to March 2019. The table includes the hourly rate for the developer, number of weeks between beginning of November 2018 and beginning

of March 2019, number of working hours per week, and the number of developers working on the project. We currently have 3 expected expenses: Labor, External software, and internal software. For some of the server image processing, we are using Google Cloud's free trial, so the cost is $0.00 until either 365 days pass or we use $300 in free credit, at which point we are charged based on our API usage.

EXPENSE	Rate / Hour	Weeks	Hours /Week	# of Dev's	Total Cost
Labor	$20.00	19	24	2	$18,240.00
External*	$0.00				$0.00
Internal**	$0.00				$0.00
TOTAL EXPENSE					**$18,240.00**
* External Software	Google Cloud - Free trial for 1 year or $300 of credit equivalent.				
** Internal Software	Any Internal software, IDE's, OS, etc. We're using free software.				

Table 1: Project Budget

Objectives / Deliverables

Table 2: Project Objectives / Deliverables Due Dates shows the project's milestones and when we plan on completing them by.

MAJOR PROJECT MILESTONES (DELIVERABLES)				
FALL OF 2018 MILESTONES				
Phase 1 Complete	10/9/2018		Team Contract Written	9/24/2018
Phase 2 Complete	10/20/2018		Project Abstract Written	10/15/2018
Phase 3 Complete	11/27/2018		User Profile Written	10/22/2018
			Elevator Speech	10/22/2018
			Fall Semester Report	11/26/2018
			Fall Presentation	11/26/2018
SPRING OF 2019 MILESTONES				
Phase 4 Complete	3/16/2019		Spring Presentation	TBD
			Tech Expo	4/10/2019

Table 2: Project Objectives / Deliverables Due Dates

Project Schedule

Figure 2: Fall Project Schedule Gantt Chart is our project schedule for Fall with the major milestones listed. ***Figure 3: Spring Project Schedule Gantt Chart*** is our project schedule for Spring with the major milestones listed.

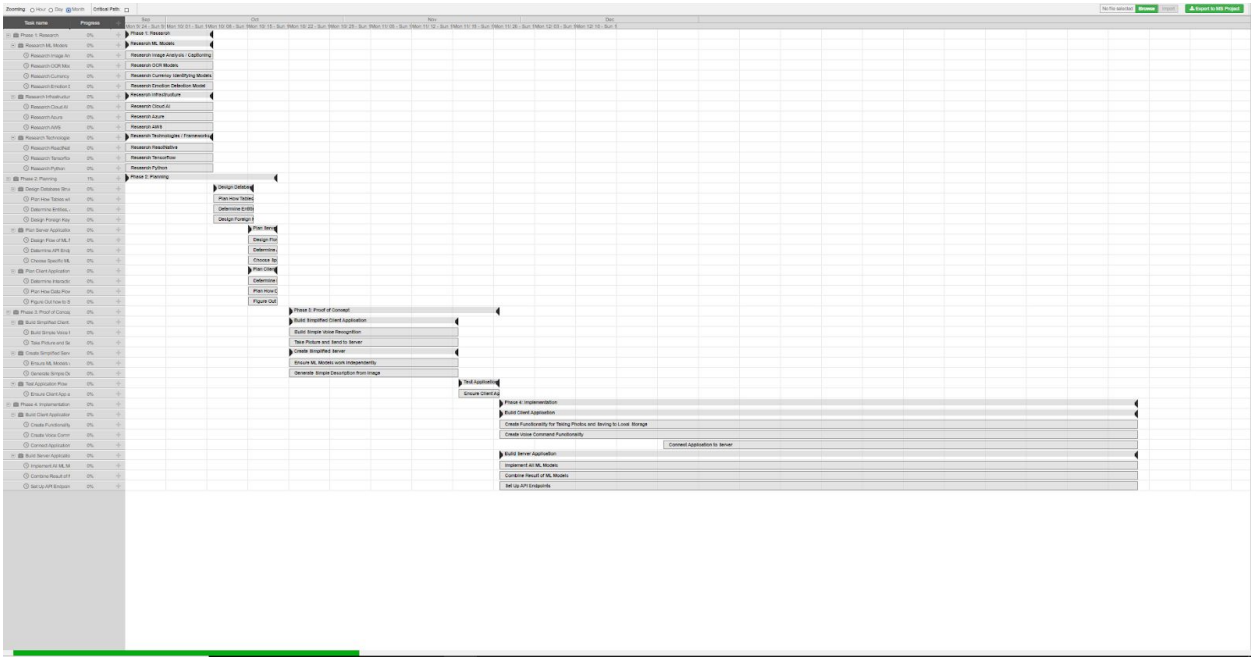

Figure 2. Fall Project Schedule Gantt Chart

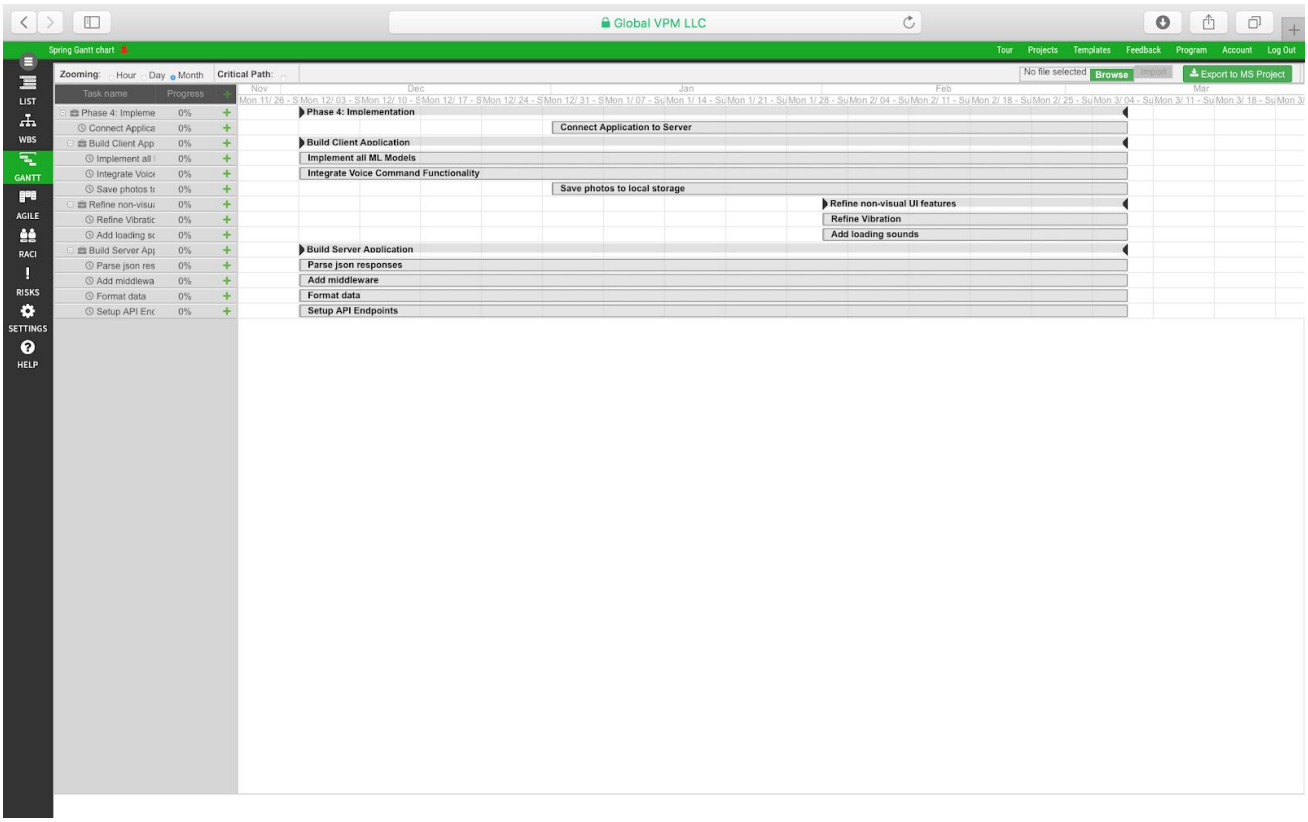

Figure 3. Spring Project Schedule Gantt Chart

TECHNICAL ELEMENTS

Network

The Iris server will be built with either JavaScript or Python and will handle the logic for processing images captured by the client application. If the server is written in JavaScript, it will utilize Node.js to run the server and expose its API endpoints with a library such as Express.js or Koa. If it is written in Python, the API endpoints will be exposed through Flask, a simple API library for Python. There will likely only be a few API endpoints with many parameters to control how the images are processed (e.g. whether the service should look for text, currency, etc.).

Application

The Iris application will be built primarily with React Native for both iOS and Android. Further native development in both Swift and Java may be necessary to improve the experience on specific mobile operating systems. The client application will communicate with the Iris server by uploading a photo taken on the user's vocal request and then returning the processed data and converting text to audible and understandable speech for output.

Security

In order to ensure that our application is secure, we plan on utilizing Transport Layer Security (TLS) technology. TLS is the modern web standard for security and is used to encrypt requests from the client to the server. A major advantage or TLS is that it works outside of our application in the transport layer, meaning no extra work needs to be done in our codebase to ensure security. Our application relies entirely on sending images to our server and images are by nature confidential. If a user were to request OCR services on a document containing private information (e.g. contact information, health records, billing, etc) we can not risk having a man-in-the-middle attack where somebody intercepts the request and see the confidential information. If we encrypted the data outside of the transport layer, there would still be a small risk that information about the user could be captured in some form. TLS, on the other hand, guarantees that the integrity of the data transfer from client to server cannot be compromised.

TECHNICAL DIAGRAM

Application Architecture

Figure 5: Application Architecture Technical Diagram shows the different logical layers and entities within our application's domain.

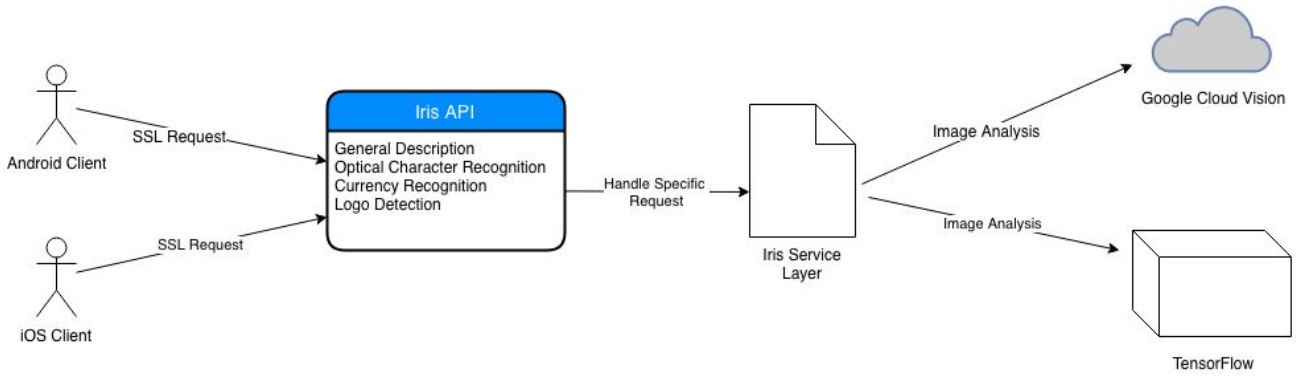

Figure 5. Application Architecture Technical Diagram

TEST PLAN

Overview

In order to ensure that Iris is a robust and high-quality application, we have put in place a series of tests. Since a visual user interface is not present in the application, we chose to focus our efforts on the user experience that Iris provides rather that any sort of user interface testing.

Scope

Iris application testing is split between Android and iOS platforms. Additionally, we chose to separate our tests into unit tests and usability tests. Unit tests are the tests that we can run to ensure the quality of our code. These tests prevent us from breaking existing functionality if we add new features or refactor the code. We utilized Facebook's Jest testing framework to write and run our unit tests. Usability tests were conducted with a real person to ensure that our application was usable and intuitive to the average user.

Objectives

All key features of the Iris application should be tested both technically by unit tests and non-technically by individuals to verify that the code is working as intended and that the application is usable. Should any part of the application not work as desired, having individual test cases allows us to identify specific issues and fix them.

Logging Test and Reporting

Bugs are logged as they are discovered, and automated unit tests are written as new functionality / features are implemented. We utilized a Node.js package called Husky to ensure that existing functionality is not broken by new changes. Husky allows us to run all our automated tests before code is pushed to version control. If a test fails, then the code will not get pushed to the repository and the failed test must be fixed. Bugs are prioritized based on their impact to the core features of the application.

Testing Procedures

Steps for testing:

1. Create multiple testing scenarios for each test case
2. Use documented steps to conduct and recreate testing scenarios
3. Record any found bugs / feedback

Unit Testing

- Given a predetermined input, do the class methods and utility functions return the expected output?
- In the event of an error, does the application behave as expected?

Usability Testing

- How does the interaction affect the user?
- Is the experience fluid and intuitive?
- Do core features function as expected in various scenarios?

Test Cases

Unit Tests:

TEST CASE	SCENARIO	INPUT	EXPECTED OUTPUT
Healthcheck	Ensure that the API is running and responding to requests	GET request to "/" or "/healthcheck"	HTTP Status Code 200
Utility Functions (10-20 small pure functions)	Ensure that all utility functions give the expected output (e.g. sort array of objects by property)	Any input that gives a known output (e.g. unsorted array)	The known output for input (e.g. sorted array)
Error Handling	An exception is thrown in the application	The thrown exception	The exception should be logged and output to the console

Usability Tests:

TEST CASE	SCENARIO	INPUT	EXPECTED OUTPUT
Voice Interaction	User tests voice recognition	**Voice Command** "Iris, test"	Iris recognizes voice input
Optical Character Recognition	User unknowingly captures photo with text in it	**Voice Command** "Iris, take a photo" "Iris, capture image" "Iris, what's in front of me?"	Iris makes note of text in the photo and asks if the user wants it read
Logo Recognition	User unknowingly captures photo that features a logo	**Voice Command** "Iris, take a photo" "Iris, capture image" "Iris, what's in front of me?"	Iris recognizes a logo and asks the user if they would like it identified
General Description of surroundings	User captures photo with no text, logos, etc.	**Voice Command** "Iris, take a photo" "Iris, capture image" "Iris, what's in front of me?"	Iris provides a description of their surroundings

Optical Character Recognition	User requests something be transcribed	**Voice Command** "Iris, read" "Iris, transcribe" "Iris, text"	Iris reads a transcription of the captured text
Logo Recognition	User requests logo identification	**Voice Command** "Iris, logo" "Iris, brand"	Iris identifies the logo for the user

Pass/Fail Criteria

All documented features identified in a test case must function as intended in each testing scenario to be deemed successful and marked as “Pass”. Additionally, the features must be considered usable and intuitive in the usability tests. Any and all unintended results will be logged as bugs, and corresponding test cases will be marked as “Fail”.

Lessons Learned

The organization of the application’s infrastructure allowed for tests to be implemented quickly and effectively. Utilizing modern development tools such as Jest and Husky allowed for more test coverage without taking too much time away from development. One area that could be improved would be the error logging aspect of the application; all errors were logged with different identifiers and timestamps to quickly find them, but aside from checking the log, there was no way to know that these errors occurred.

CONCLUSION

Fall Semester 2018

The Fall Semester of 2018 has primarily been spent on Phases 1-3 (Research, Planning, Proof of Concept). We determined our project focus, how to accomplish it, and built prerequisites to the full functionality of the application. We also began experimenting with the technologies we will be using for this application and managed to build a prototype that is capable of providing basic descriptions to the user. The work we have completed in the Fall Semester was a significant step in the right direction and will help pave the way for

improvements, iterations, and finally some polishing of the application.

Spring Semester 2019

The Spring Semester of 2019 will be spent on Phase 4: Implementation, our final deliverables, fully-functional iOS and Android applications, free, open-source, and available for download. The application should accomplish all of the minimum-viable deliverables and hopefully some of our stretch goals. We will have a significant amount of work to complete as far as the application's functionality is concerned, but the bulk of the work could potentially be in the application's infrastructure. Currently, the client prototype is compiled through Expo and does not actually generate a standalone Android application file or iOS application file. The server prototype runs locally off of one of our laptops or desktops. In their current state, the application is not scalable or easily distributed. In order to reach that goal, we will need to compile the final application to standalone iOS and Android apps and find a way to distribute them. Additionally, we will need to acquire some sort of public hosting (AWS, free online hosting, open personal computer to public traffic) so that users are able to use the application.

APPENDIX A. ADDITIONAL INFORMATION

The team contacted the following professionals for clarification and additional insight.

1. Abdou Fall, Advisor

APPENDIX B. REFERENCES

[1]	"React Native Documentation," 2018 [Online]. Available: http://facebook.github.io/react-native/docs/getting-started.
[2]	"Google Cloud Vision," 2018. [Online]. Available: https://cloud.google.com/vision/docs/.
[3]	"React-Native-Voice," 2018. [Online]. Available: https://www.npmjs.com/package/react-native-voice.
[4]	"Flask (A Python Microframework)," 2018. [Online]. Available: http://flask.pocoo.org/.
[5]	"Node.js," 2018. [Online]. Available: https://nodejs.org/en/.