uC[≡]de

uCode - Run Handwritten Code Anywhere

Dr. Chen: Senior Project I

Team Names

Instructor Comments/Evaluation

Table of Contents

Abstract

This paper is designed to be read by the software developers of uCode. It will lay out the development side designs and requirements of this application. The application will scan handwritten code and compile it on a backend server, sending prompts and displayed lines of text back to the phone. The developers can use the information provided in this document to gain insight on their best process to handle the project. The details list the interactions between modules of the code, as well as what programming language will be used.

Description of the Document

Purpose and Use:

The purpose of this document is to give the software development team overall guidance to the architecture of the software project. This document will also show in detail the programming language and functions needed to make the project work. It will be used by the software development team as a way to keep the project on track without going into scope creep or going down the wrong path.

Ties to the specification document:

This document simply extends what the specification document shows and adds in components relevant to the software development side. This document will show what programming language will be used and what style of programming architecture will be implemented into the project.

Intended Audience:

This document's primary audience are software developers and designers due to the technical nature and verbage within. This document is not intended for the client, but it may be in their best interest to have a technically capable third party member in order to review the design documentation for conformity to the client's requests. Software developers and designers should use this document to better understand the scope of uCode, including but not limited to the application's modules, data flow, variables, functionality, architecture and system requirements.

Project Block Diagram w/ Description

The uCode project will consist of two parts: Frontend code that resides on a smartphone (either Android or iOS), and backend code on a Linux or Windows server. The phone will interact with the backend server by sending it the desired image the user selects. Once the backend has received the image, it will then process the image and return the text back to the client device. From there the client device can interact with the code while it is being executed.

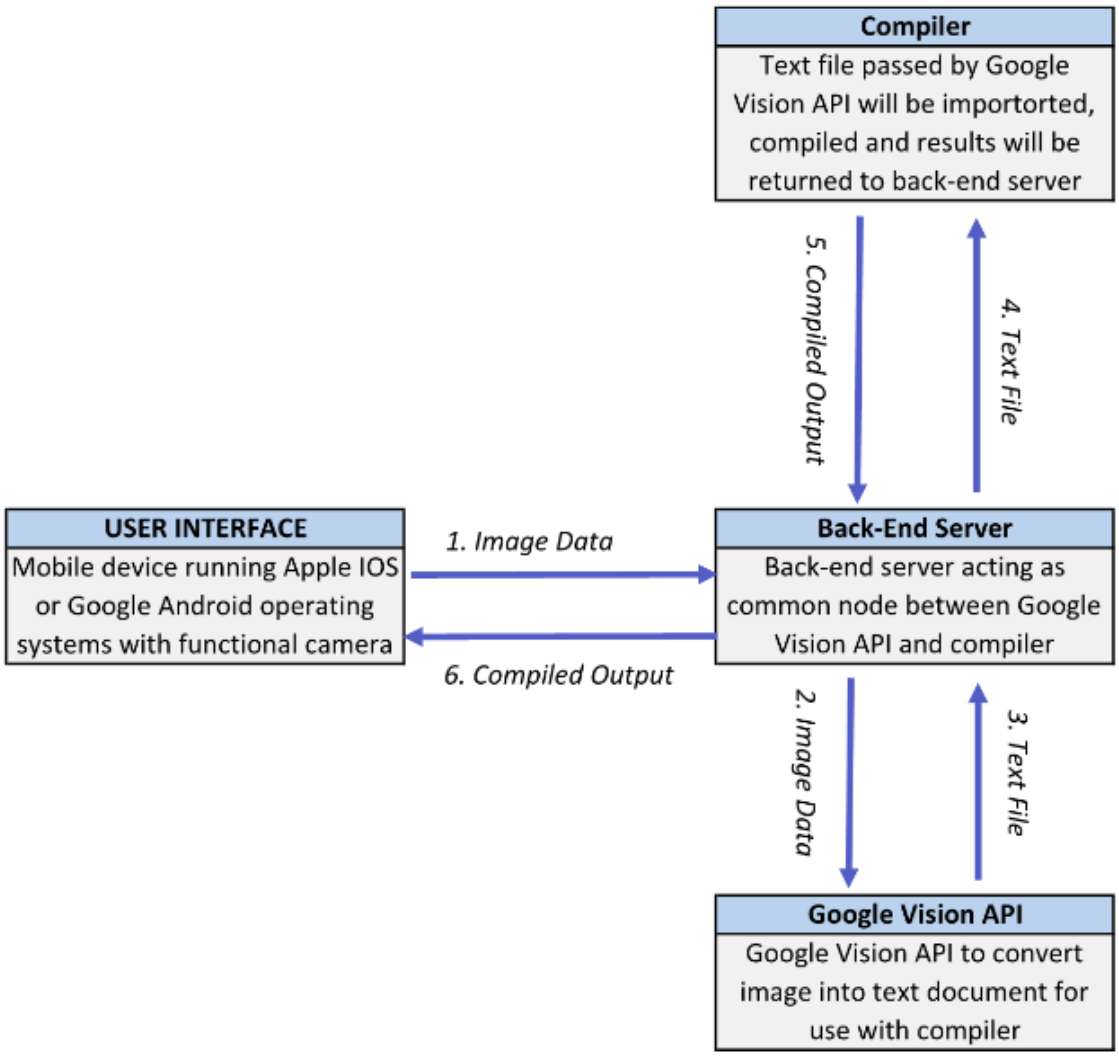

Figure 1: Block Diagram

Design Details

System Modules and responsibilities:

- Architectural Diagram

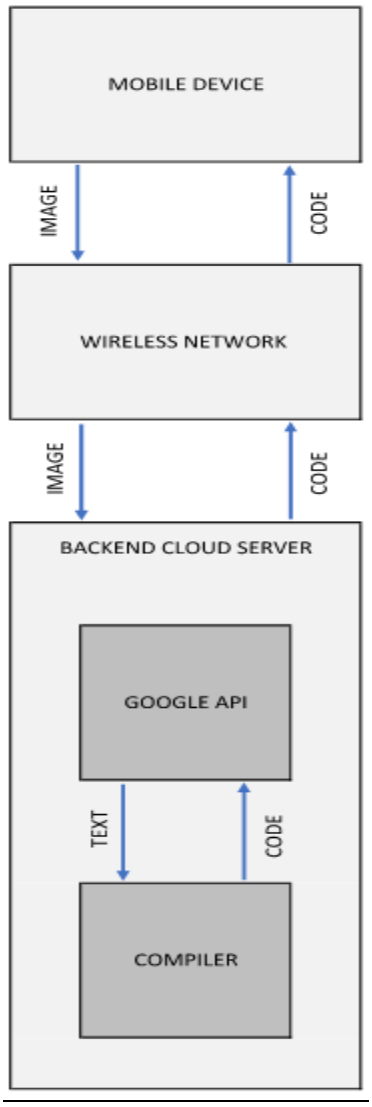

Figure 2: Architectural Diagram

Figure 2 Description: Data will be generated by the end user and transmitted by the mobile device. The image data will be relayed over a wireless network and received by

the backend server. The backend server contains both the Google API and compiler that are tasked with converting the image data into compiled code (Vision API, 1). The compiled code will be returned to the mobile device over the wireless network.

- Module Cohesion

  Our project expresses module cohesion with functions expressed in the Project class. The Project class contains all the features necessary to manage a user's projects. They can add, delete, save, and load. Having all the project functions in one properly-named class makes the code readable and logical.

  All the View classes contain the code needed to handle the interface. They are all independent of each other. Views can be added or removed with ease to develop new interfaces without touching the underlying Controller classes (marked in the design details as control classes).

- Module Coupling

  For the cell phone, there are various types of View classes that will be utilized together in order to accomplish different screens. The CameraView, EditorView, and ExecutionView classes are all related to each other, but each has their own specific purpose. The children of these views are the controller classes, which respond to the actions the user requests. The phone app will be using the MVC (Model View Controller) system. This is a common system used in modern GUI applications, as it "allows flexible structures" where the interface and application functions are independent ("Graphical user interface", 2018).

The backend program will express module coupling with the Recognizer, Compiler, and Executor classes. These classes will process in order and rely on each other to successfully run a project.

Design Analysis

- Data Flow

  The dataflow of uCode is handled by the payload function that either passes the image or a string of text.The user activates the application and sends their picture to the backend server. The data is processed by machine learning and the compiler where it generates the executable for the program. The executable is ran and the data from the program is sent back to the user.

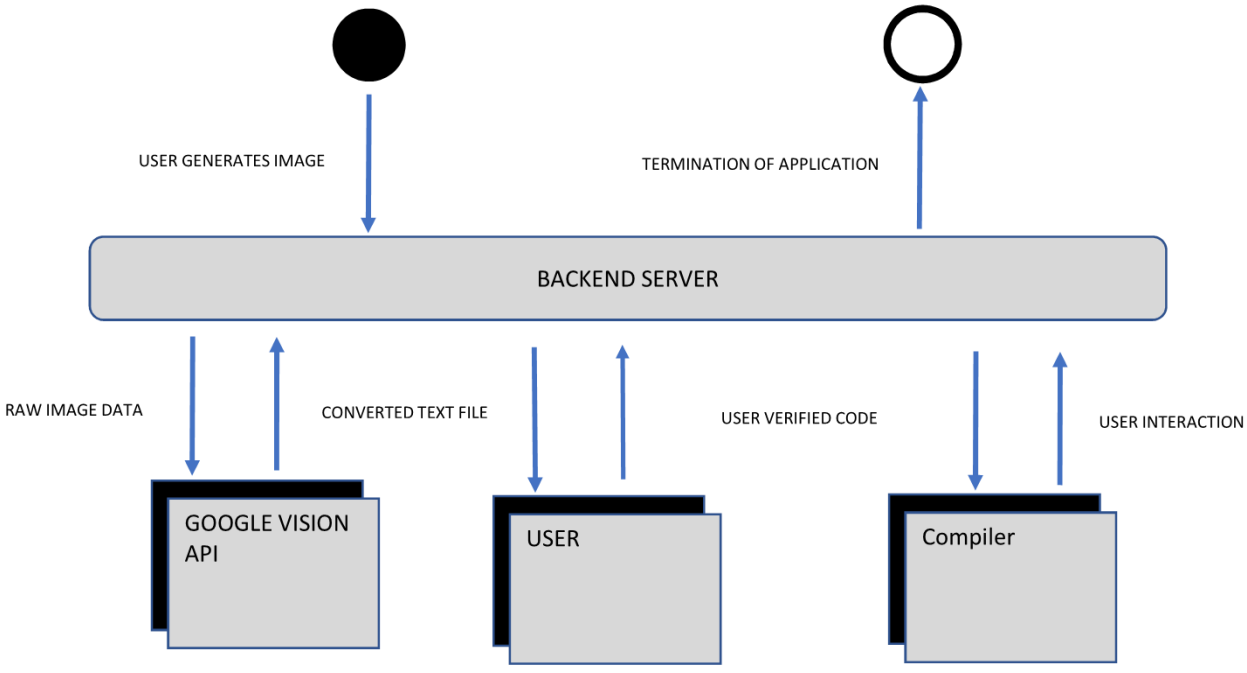

Figure 3: Data Flow Diagram

Design Organization (Object Oriented Design)

Detailed tabular description of Classes/Objects

Class name: CameraView

Class description: The Camera View class is a boundary object used to show the camera on the cell phone. This class handles the interaction between the user and the camera.

Class data members: Camera Object, View Object

Class member functions: showCameraPane(), hideCameraPane(), showImage()

Class name: EditorView

Class description: The Editor View class is a boundary object used to show the editor page on the cell phone. This allows the user to modify the code after the image has been converted into digital text.

Class data members: View Object, TextArea Object

Class member functions: showEditorPane(), hideEditorPane()

Class name: ExecutionView

Class description: The Execution View class is a boundary object used to interact with the program as it is being executed. It has functions to handle the addition of messages. Every message that is added is displayed whether it was from the client or the server program executing.

Class data members: View Object, TextArea Object

Class member functions: showExecutionPane(), closeExecutionPane(), addMessage()

Class name: Recognizer

Class description: The Recognizer Class is a control class that handles the interaction between the machine learning algorithm API and the backend program. It has the ability to send an Image file and receive text back.

Class data members: String encodedImage

Class member functions: recognizeImage(), sendToRecognizer(), getFromRecognizer()

Class name: Compiler

Class description: The Compiler Class is a control class that sends the processed code to the compiler. This will compile the code and return any errors in the payload if there is a compilation problem.

Class data members: String filename

Class member functions: sendToCompiler(), receiveFromCompiler()

Class name: Executor

Class description: The Execution Class is a control class on the backend that executes the code that was just compiled. It runs the code and then sends messages with the payload object. It also interacts with the frontend to allow interactivity.

Class data members: String executableFileName

Class member functions: sendMessage(), receiveMessage(), runCode()

Class name: Camera

Class description: The Camera class is the controller class that manages the actual image taking process. Once the user initiates an image being taken in the CameraView class, the Camera class will start its processes.

Class data members: Camera object (from smartphone libraries)

Class member functions: takePicture(), deletePicture()

Class name: Editor

Class description: The Editor class is the controller class that handles the fetching and sending of the code in the Editor. When the user asks to submit the code, the Editor class will be executed after the EditorView calls the main editor class.

Class data members: String object

Class member functions: editCode(), submitCode()

Class name: Execution

Class description: The Execution class is the controller that handles the message in/out process of the execution phase. Once the user presses enter the corresponding view object, the Execution class will send a message to the backend server. When the server sends text back, the Execution class puts the message back on the screen.

Class data members: String object

Class member functions: sendMessage(), receiveMessage()

Class name: ImagePayload

Class description: ImagePayload will contain the client ID, session ID and the image itself.

Class data members: Image object, String object

Class member functions: imgImage(), sessionIdInteger(), codeString()

Class name: TextPayload

Class description: TextPayload will contain the client ID, session ID and the string being sent between the phone and the server.

Class data members: String objects

Class member functions: messageString(), responseString()

Functional descriptions

CameraView Class

showCameraPane()

Input:

The showCameraPane() function has no input.

Output:

The function will output the camera pane to the screen.

Return Parameters:

The only values returned are any exceptions thrown during error checking.

Types:

The data types used within this function are based on the built-in Android and iOS functions required to change panes.

hideCameraPane()

Input:

The hideCameraPane() function has no input.

Output:

The function will hide the camera pane and return to the previous screen.

Return Parameters:

The only values returned are any exceptions thrown during error checking.

Types:

The data types used within this function are based on the built-in Android and iOS functions required to change panes.

showImage()

Input:

The image file that is to be displayed within the frontend GUI.

Output:

Displays the image file that was received from the passing function.

Return Parameters:

The only values returned are any exceptions thrown during error checking.

Types:

Strings

EditorView Class

showEditorPane()

Input:

The showEditorPane() function has no input.

Output:

The output of this function will display the editor pane on the screen.

Return Parameters:

The only values returned are any exceptions thrown during error checking.

Types:

The data types used within this function are based on the built-in Android and iOS functions required to change panes and boolean flags for error checking.

hideEditorPane()

Input:

The hideEditorPane() function has no input.

Output:

Thethis function will hide the editor pane on the GUI.

Return Parameters:

The only values returned are any exceptions thrown during error checking.

Types:

The data types used within this function are based on the built-in Android and iOS functions required to change panes and boolean flags for error checking.

ExecutionView Class

showExecutionPane()

Input:

The showExecutionPane() function has no input.

Output:

The mobile device should display the execution pane.

Return Parameters:

This function will return the success or failure of the GUI displaying the execution pane .

Types:

The data types used within this function are based on the built-in Android and iOS functions required to change panes and boolean flags for error checking.

closeExecutionPane()

Input:

The closeExecutionPane() function has no input.

Output:

The mobile device should closes the execution pane.

Return Parameters:

This function will return the success or failure of the execution pane displaying.

Types:

The data types used within this function are based on the built-in Android and iOS functions required to change panes and boolean flags for error checking

Recognizer Class

recognizeImage()

Input:

The recognizeImage() function takes a string path to file, imageFile.

Output:

The recognizeImage() pulls the image file as a string and calls sendToRecognizer() to send to the server and getFromRecognizer() to listen for a response from the server.

Return Parameters:

This function will return success or failure of the recognition procedure, received by the server, and any other error checking.

Types:

The data types used within recognizeImage() are string, integer and boolean data types.

sendToRecognizer()

Input:

sendToReconnizer() will receive the image, iString, to send to the server.

Output:

The sendToRecognizer() function does not provide output.

Return Parameters:

sendToRecognizer() returns the success or failure code of the function.

Types:

The sendToRecognizer() function will require string, integer and boolean data types.

getFromRecognizer()

Input:

No input is needed for the getFromRecognizer() function.

Output:

There is no output for the getFromRecognizer() function.

Return Parameters:

This function will return if the recognition and transmission was successful or not and any other error checking.

Types:

The data types used within this function include string, integer and boolean.

Compiler Class

sendToCompiler()

Input:

The sendToCompiler() function will receive txtFile.

Output:

This function will have no output.

Return Parameters:

The function will only return flags for error checking.

Types:

The data types used within this function will be string, integer and boolean.

receiveFromCompiler()

Input:

No input is required for this function.

Output:

There will be no output from this function.

Return Parameters:

This function will return the success or failure of the text to compile and any debugging error messages from the compiler, the compiled executable or an error checking flag.

Types:

The data types used within this function are string, integer and boolean.

Project Class

newProject()

Input:

newProject() requires projectName and newImage.

Output:

newProject creates a new project object under the projectName identifier.

Return Parameters:

newProject may return exceptions thrown during error checking.

Types:

The data types within this function are string and boolean.

saveProject()

Input:

This function requires projectName, fileLocation and txtFile.

Output:

saveProject will output a success/failure message to the user.

Return Parameters:

saveProject will return any error checking exceptions.

Types:

The data types used within the saveProject() function will include integer, string, char and boolean.

loadProject()

Input:

This function will require the fileLocation and projectName.

Output:

The function will output a read-only text file from the projectName identifier.

Return Parameters:

The function will return any error checking exceptions.

Types:

The data types used within loadProject() include string and boolean.

editProject()

Input:

The editProject() function only requires the text file.

Output:

This function will output a text editor with the text from text file with read/write permissions.

Return Parameters:

This function will return any error checking exceptions.

Types:

The data types used within this function include string, character, integer and boolean.

runProject()

Input:

runProject will require a text file

Output:

runProject will call functions within the executor class to compile txtFile and output the executable code.

Return Parameters:

This function will return any error checking exceptions.

Types:

The data types used within this function include string and boolean.

Executor

sendMessage()

Input:

This function receives a text file, compileResults.

Output:

sendMessage() sends compileResults as a message to the receiveMessage() function listening on the mobile device.

Return Parameters:

This function will return the success or failure of a message to be sent or any other error checking.

Types:

The data types used within this function will include string, integer and boolean.

receiveMessage()

Input:

No input is need for the receiveMessage() function.

Output:

No output comes from the receiveMessage() function.

Return Parameters:

This function returns the message msgRecieved or any error checking.

Types:

The data types used within this function includes string, integer and boolean.

runCode()

Input:

No input is needed for the runCode() function

Output:

Runs the code that was sent to the server

Return Parameters:

Returns the long string out outputted code from the execution phase (if there is no interaction)

Types:

No data types, just a command sent to the backend

Camera

takePicture()

Input:

This function receives the image, iString.

Output:

This function will display the image taken.

Return Parameters:

takePicture() will return any error checking within the function.

Types:

The function will use string and boolean data types.

deletePicture()

Input:

deletePicture() will receive the file path, imageFile.

Output:

There will be no output for this function.

Return Parameters:

The function will return any error checking done within.

Types:

The data types used in deletePicture() will be string and boolean.

Editor

editCode()

Input:

No arguments are needed for the editCode() function.

Output:

This will call the showEditorPane() from the EditorView class where the keyboard is brought up and the code can be edited.

Return Parameters:

editCode() returns the success or failure of the process of editing the code and any other error checking.

Types:

This function will use string and boolean data types within.

submitCode()

Input:

submitCode() will receive the finished code file, txtFile.

Output:

There is no output for this function.

Return Parameters:

This function will return any error checking done within this function.

Types:

The data types used within this function will include integer, string and boolean.

Files accessed:

The phone will store project files that include the source code they compiled. The files stored on the server will only be temporary. The server will write out a source code file whenever it is passed a string of source code. It will use this file to then execute and run.

Real-time requirements:

The program needs to run in a reasonably speedy amount of time. If the application is too slow to recognize input from the image, the user may be frustrated and leave the platform. In addition, the interaction with live execution is important, as the response times between input and output need to be quick. Ideally, it would be as fast as if the interaction was on a local system.

Messages:

The messages sent between the frontend and backend processes will include the following pieces of data:

Message Type	**Source / Destination**	**Data**
sendMessage()	Phone <-> Backend	Interaction IO String
receiveMessage()	Phone <-> Backend	Interaction IO String
sendToCompiler()	Phone -> Backend	Source Code String
receiveFromCompiler()	Backend -> Phone	Compilation Results String
sendToRecognizer()	Phone -> Backend	Image from phone
receiveFromRecognizer()	Backend -> Phone	Recognized source String
recognizeImage()	API <-> Backend	Image loaded into backend from user's phone is sent to the machine learning OCR algorithm

Narrative / PDL:

Main Menu:

1. New project
2. Load project

New Project:

1. Name project

   Choose project name & move on to step 2

2. Take a photo

   ShowCamera:

   a. Shutter button

      Satisfied?

         No? Clear and retake image

         Yes? Recognize image and send text to the editor. Go to editor(3)

3. Load photo from disk

   Satisfied?

      No? Clear and reselect an image from disk

      Yes? Recognize image and send text to the editor. Go to editor(3)

4. Go to editor

   showEditor

      Allow code to be edited

      Run code?

         Yes? Send to compiler and executor classes

         No? Do nothing

      Code is running

         Interaction?

            No? Complete program and display output

            Yes? Have user interact with program as it is being executed

         Take user back to editor once compilation and execution has completed so they may revise their code or move to start a new project

Decision: Programming language / reuse/ portability:

Our backend code will be written in Python. Python allows for the rapid-prototyping and has libraries for various API's and functions. By using Python, we would write less code that doesn't directly pertain to our project, such as serializing an image over the network. Python "enables clear programming on both small and large scales" (Python 1).

Implementation Timeline

The following is a timeline of implementation with incremental deadlines designed to keep the project on track.

Activity	Jan	Feb	March	April	May	May+
Component Level Design						
Software Design						
Unit Testing						
Subsystem Integration and Verification						
System Integration and Verification						
System Validation						
Operation and Maintenance Manuals						
Changes, Upgrades and Retirement						

Design Testing

Design testing will be done at regular intervals throughout the project's development. After each part of the project is completed, like the camera app or the network to the backend server, it will be tested alone during development and together with the project's other modules to ensure the build is sound. Due to the project's initial data needing to be handwritten, time could be saved by using several distinct samples of varying degrees of quality that can be tested. The samples will also have programs of varying degrees of complexity for testing purposes. Once the developers create the part where the backend server will send responses asking for user input, the test programs can change to just ones with multiple levels of user input. We will be able to see how the project handles good to bad quality data as we start the build, and then simple to complex programs as the build is finished. The testing team will be the development team. The quick real time data we can get back if our own team tests the data, will be better than if we outsource it to volunteers.

References

Beal, V. (n.d.). API - application program interface. Retrieved October, 2018, from https://www.webopedia.com/TERM/A/API.html

Beal, V. (n.d.). TCP - Transmission Control Protocol. Retrieved October, 2018, from https://www.webopedia.com/TERM/T/TCP.html

Graphical user interface. (2018, October 19). Retrieved from https://en.wikipedia.org/wiki/Graphical_user_interface

Gazarov, P. (2016, August 13). What is an API? In English, please. – freeCodeCamp.org. Retrieved from https://medium.freecodecamp.org/what-is-an-api-in-english-please-b880a3214a82

NPR Staff (2013, March 03). Teaching 2.0: Is Tech In The Classroom Worth The Cost? Retrieved from http://www.npr.org/2013/03/03/173372736/teaching-2-0-is-tech-in-the-classroom-worth-the-cost

OCR Technology. (2018, November 11). Retrieved from https://www.abbyy.com/en-us/finereader/what-is-ocr/

Oracle. (n.d.). What Is a Socket? Retrieved from https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

Python (programming language). (2018, November 02). Retrieved from https://en.wikipedia.org/wiki/Python_(programming_language)

Rouse, M., Gillis, A., & Silverthorne, V. (2018, September). What is integrated development environment (IDE)? - Definition from WhatIs.com. Retrieved from https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment

Vision API - Image Content Analysis | Cloud Vision API | Google Cloud. (n.d.). Retrieved October, 2018, from https://cloud.google.com/vision/

What is Bluetooth? - Definition from Techopedia. (2018, November). Retrieved from https://www.techopedia.com/definition/26198/bluetooth

Appendix A: Technical Glossary

API (Application Program Interface): a set of routines, protocols, and tools for building software applications (Beal, 2018).

Android: a mobile operating system developed by Google.

Backend server: cloud-based Linux server.

Bluetooth: an open wireless technology standard for transmitting fixed and mobile electronic device data over short distances (What is Bluetooth?, 2018).

CentOS: A popular distribution of the Linux operating system.

Client: Computer hardware or software that accesses a service made available on another computing device.

Compiler: a program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer.

Frontend: A term commonly used in this document to describe the user-side application also known as the graphical user interface (GUI)

Functional requirements: basic functions completed by the application.

Google Vision: API used to access Google's image recognition capabilities. "Cloud Vision offers both pretrained models via an API and the ability to build custom models using AutoML Vision to provide flexibility depending on your use case" (Google Vision, 2018)

GUI (Graphical User Interface): a form of user interface that allows users to interact with electronic devices through graphical icons and visual indicators (Graphical User Interface, 2018)

IDE (Integrated Development Environment): "a software suite that consolidates basic tools required to write and test software." (Rouse, M. et. al., 2018)

iOS: a mobile operating system created and developed by Apple.

Machine Learning: a branch of artificial intelligence where systems can learn from data, identify patterns and make decisions with minimal human intervention

Networking: a process that cultivates information and data on a digital telecommunications network and disperses it over data connections to different nodes

Node: a point on a computer network (e.g. cell phone, server, computer, printer, etc.)

Nonfunctional requirements: tools used to complete the basic functions within the application.

OCR Technology: A technology that enables you to convert different types of documents, such as scanned documents, PDF files or images captured by digital camera into editable and searchable data (ABBY, 2018).

Python: an interpreted high-level programming language for general-purpose programming (Python (programming language), 2018).

Sockets: endpoint of a two-way communication link between two programs running on the network (Oracle, 2018)

TCP connection: one of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent (Beal, 2018).

Ubuntu: A popular distribution of the Linux operating system.

Appendix B: Team Details

The requirement document was developed by the following individuals, all of whom contributed the following:

Appendix C: Workflow Authentication

I, xxxxx, attest that I executed the functions listed within the team details section of the document. Also, I agree with all aforementioned information stated within the requirements document.

_____	_____	_____
Printed Name	Signature	Date

Appendix D: Report from the Writing Center

Cal U Writing Center Report

Client:

Staff or Resource: Brittany K.

Date: December 6, 2018, 11:00am - 12:00pm