

Kyle Venenga

VCAD Project Documentation



Executive Summary

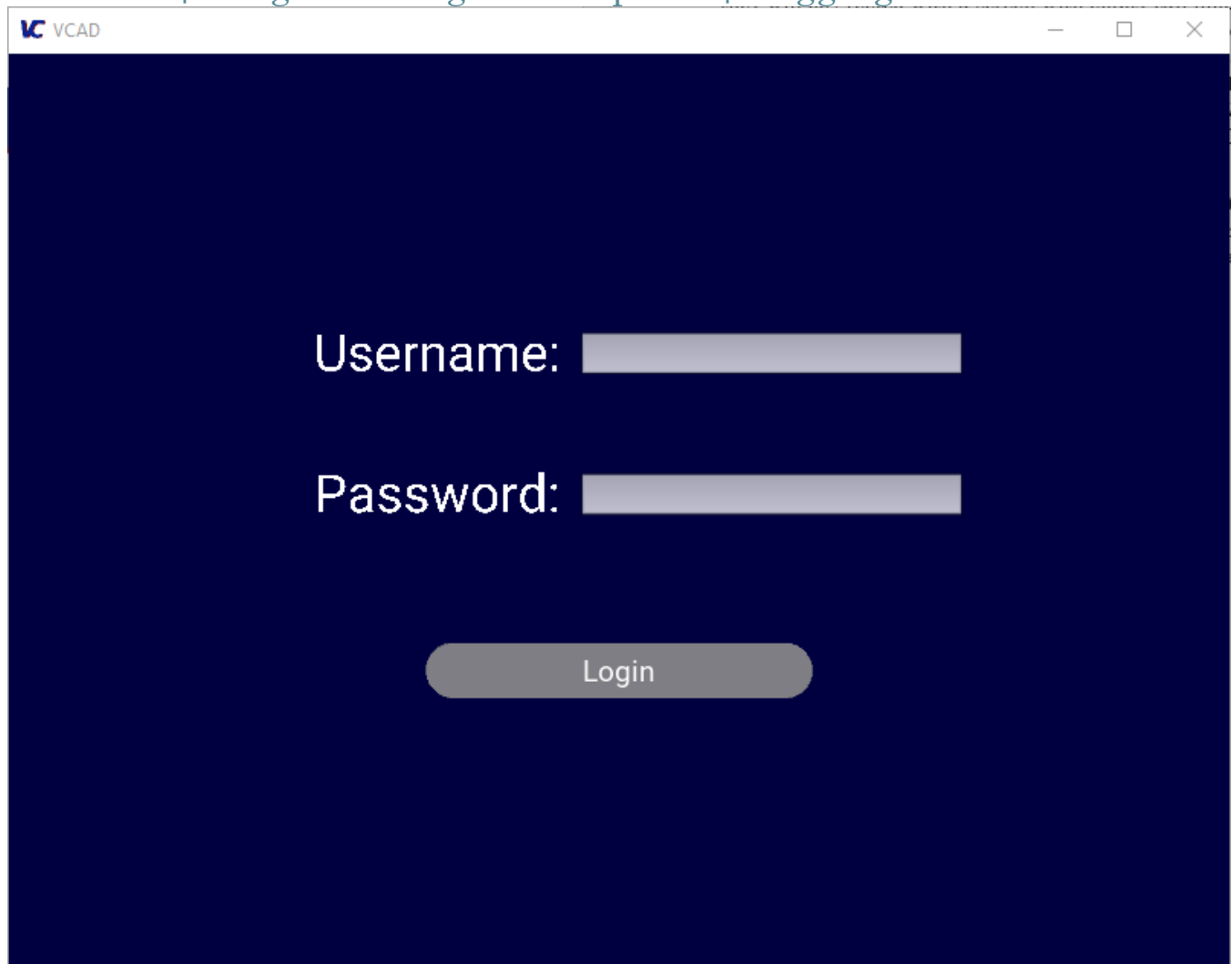
Police use computers in their squad cars to look up names, plates, write reports and tickets, and view call information. These systems are called a CAD (Computer Aided Dispatch) that allow for the ease of their patrol, and help limit radio traffic as much as possible. These systems also allow dispatchers to send calls to their officers, and view which officers are available and busy. Police CAD's allow the dispatcher the ability to see the location of their officers as well, to best provide a quick response to call locations.

VCAD will be a CAD system that is developed for the use of a security company, not the police. My brother is a patrol officer for a security company, and responds to break ins where the police cannot. They currently use computers for report writing, however they do not currently have a CAD system and rely heavily on radio communication. A CAD system for a security company doesn't need to lookup people or vehicles and write tickets. It strictly needs to be able to send and receive call information, write reports, store this information in a database, as well as allow the dispatcher the ability to see who is available and who is not.

The main feature of this CAD system would be the following; First the officer would need to be able to login to the system using their credentials. Their computer would likely be using a virtual machine connecting back to a server at the office, to eliminate data transfer of sensitive material. As soon as the officer logs in, they will be greeted with a screen with empty call information, such as address, phone, place, city, zip, description and call type. They could then write a report right from another screen at the click of a button. As soon as a call comes in, the officer will be marked as busy for both the officer and the dispatcher, there will be two buttons on the top for busy and available. The ability to also mark when an officer arrives on scene is a must, as it shows response time to clients. When a call comes in to the officer the computer will read text-to-speech the call information out loud. Finally, the officer can view and write reports for all of their previous calls, in a nice multipage section.

The dispatcher CAD system will show all officers and their availability. They will be able to submit a new call, enter the information, and send it to an officer. The dispatcher will be able to mark officers busy, on scene and available. There is error handling within the dispatcher screen, allowing the dispatcher a smooth experience when using the system.

Scenario | Program Usage Description | Logging in

A screenshot of a web application window titled "VCAD". The window has a dark blue background. In the center, there are two input fields: "Username:" followed by a light gray rectangular box, and "Password:" followed by a light gray rectangular box. Below these fields is a rounded rectangular button with the text "Login". The window's title bar includes the "VCAD" logo and standard minimize, maximize, and close buttons.

When a user wants to login to their screen, they will be greeted with a login screen. The best part about this login screen is that it redirects the user to the page they are responsible for. If Tom is a dispatcher, he will come to this screen and login, he will be directly taken to the dispatcher screen rather than the officer screen.

Scenario | Program Usage Description | Dispatching



The screenshot displays a web application interface for a dispatcher named Paterson. The interface is divided into two main sections: 'Create Call' on the left and 'Online Officers' on the right.

Create Call Section:

- Call Type:** A text input field.
- Street Address:** A text input field.
- City:** A text input field.
- Zip Code:** A text input field.
- Place:** A text input field.
- Phone:** A text input field.
- Description:** A large text area for additional details.

Online Officers Section:

- Prev** and **Next** buttons for navigating between pages of officers.
- Page: 1** indicates the current page.
- Officer List:** A list of officers with their status indicators. The first officer, Venenga 1001, is shown with three status buttons: **10-23** (blue), **10-7** (red), and **10-8** (green). The **10-8** button is currently selected.
- Send** button: A button to send the call to the selected officer.

Tom is a dispatcher, and he needs to be able to monitor all the active officers, and be able to send them calls. When he logs in, he is greeted with a dispatcher screen such as this. He can see there is currently one officer online that is currently available for a call. He then can type out the information into the fields, and hit send. As soon as he does this, the officer will then switch to 10-7, busy. When the officer calls in that he is on scene, Tom can now mark him 10-23. When the officer tells them that they are free, he can mark them back 10-8, which will automatically mark them off scene and available, both on his screen and the officers.

Scenario | Program Usage Description | Officer

Current Call		Previous Calls	
Call Type:		13	Aug 10 11:18 PM
Street Address:		12	Aug 10 08:49 PM
City:		11	Aug 10 08:49 PM
Zip Code:		10	Aug 10 05:50 PM
Place:		9	Aug 10 05:49 PM
Phone:		8	Aug 10 05:47 PM
Description:		7	Aug 10 05:46 PM

Call ID	Time	Address	Action
13	Aug 10 11:18 PM	123 N. Walworth Dr.	Report
12	Aug 10 08:49 PM	123 N. Walworth Dr.	Report
11	Aug 10 08:49 PM	123 N. Walworth Dr.	Report
10	Aug 10 05:50 PM	535 Worthington Av.	Report
9	Aug 10 05:49 PM	432 Westwood Ct.	Report
8	Aug 10 05:47 PM	645 E. Malta Ave.	Report
7	Aug 10 05:46 PM	123 N. Walworth Dr.	Report
6	Aug 10 05:45 PM	123 N. Westword St.	Report
5	Aug 10 05:44 PM	645 W. Belview Blvd.	Report
4	Aug 10 05:43 PM	563 N. Acer Ct.	Report

Venenga is an officer, when he logs in this is the screen he sees. He has been on many other calls, as he can tell by the right side of the screen. If he needs to update the reports, he can click on the report button associated with the address, time and call ID. When he writes and submits the report, it updates the information in the database. Right now, Venenga is waiting for a call. When one does come in, the 10-7 button will light up, and 10-8 will deactivate, as well as a message from the computer reading him call information out loud. Venenga can read for himself the call information in a nice, night friendly GUI. Venenga can mark himself on scene and finished with the call with the top buttons. This will notify the dispatcher without making noise on the radio.

System Architecture

There is an overall database that would live on a server, and likely also run many virtual machines out to the users to keep data in one spot. VCAD would live on the virtual machine, allowing access to open it, and connect locally to the database for security purposes. An account can be logged into as many times as they would please, due to the fact that eventually this may become a mobile application to allow for foot patrol, as well as in-car, in-office and any other area of access. There isn't anything major that could cause a problem due to this type of architecture. Whenever a change is made to the database, the latest change is the one that will stick. If the officer were to write a report and somehow hit submit on two different instances, whichever instance was submitted last would take. All of the code runs an application on Windows machine currently, however it was built with Kivy allowing it to easily be transferred to any operating system, including mobile, which is why I chose Kivy. The code itself talks to the database, and logs in to an account that is restricted to viewing, and updating table contents, not the database or the table information themselves, this allows for no errors whatsoever to destroy the database.

Source Code Structure

Code Directory	
Directory	Usage
Main VCAD Directory	Holds all of the python files, as well as other directories
Audio	Holds audio files associated with the program
Kivy Files	Holds all of the GUI files
Database File	Just holds an export of the database so anyone can start a server with it
Build	Holds all of the built content/packaged, exe, for Windows.

Highlighted rows indicate directories containing source code.

Executables

There is currently one executable, vcad.exe, that launches the program. From there the users can access the different screens associated to their account. Currently the executable only runs on Windows, but has the ability to easily be ported to other operating systems.

vcad (vcad.exe)

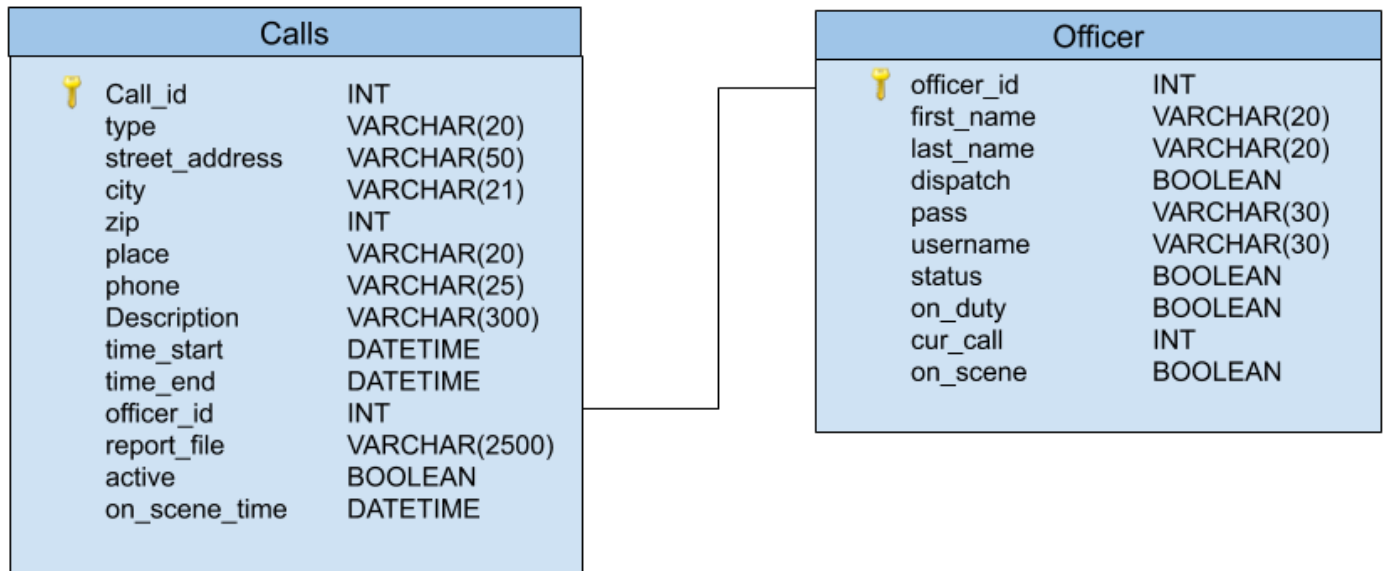
Launches the program.

Code Architecture

Code is using three main components. Python for logic base, SQL for database, and Kivy for the GUI. All three of these work together and are dependent on one another for VCAD to work.

Database or Data Store

MySQL database hosted on a local server. Two tables, one to hold call information, the other to hold officer information. Simplistic design, but effective for the software.



Programming Language | Python, Kivy, SQL

Python

The heart of VCAD is written in Python using the Kivy framework for the GUI. The code was written in the PyCharm IDE on Windows 10. There were many tools used/imported for this project which I will talk about below.

Kivy

Kivy is used to allow cross-platform support for GUI using Python, and can compile into other languages meant for operating systems such as Android. Kivy has a HTML/CSS style of GUI coding, that takes a Python spin. There are no brackets, or anything that CSS uses for formatting, it's much like Python as it uses indentation. Everything indented in belongs to the parent, much like Python. Documentation was kind of rough in many aspects of the development, however it was quite easy to learn once the baseline information was figured out. Kivy allows you to build a GUI using Python code or the Kivy code, and uses ID's to access a widget/item in Python.

Each widget was created as a class, and any logic and functions could be written within the class. Such as buttons logic when pressed.

SQL

SQL (MySQL) is the meat of the program. Two basic tables running on a database server are utilized in this program. Most of the queries were written within Python using PyMySQL.

Tools

Threading

Threading is a huge part of my project. I have two major threads that are used. One for each of the main screens, dispatcher and officer. These threads run through the duration of the screens life, and do make database checks to update the screen information. This is how the dynamic aspect of the software works, a constant check and update.

Kivy GUI

There are many different tools with Kivy, all the different GUI tools, such as geometry, labels, buttons, etc. All of these are imported and utilized throughout the program. If a button is initialized in a .kv file then the item does not need to be imported. However, if a button is created within the main python file, it does.

GTTS / Google Translator

Google text to speech and translator are utilized within the project to read the information out loud to the officer. I have it coded that it will read St. as street, av. As avenue, and so on. This allows a smooth, and genuine experience for the officer, while allowing the dispatcher to abbreviate. Text to speech allows the officer to keep eyes on the road at all times, rather than looking away to read a call on a screen.

PyMySQL

This tool allows access to SQL connection objects, cursor objects and allows the program to interact with a database. This is the core functionality of my program, and nothing works without access to the database.

PyGame

PyGame is powerful, and has a lot within its library. I used it to play a sound out loud.

Other Tools: DateTime, Time, OS, tempfile, playsound

Project Classes

Classes within the project are used to abstract re-usable pieces of code. Classes are also used to group related values, known as properties. The project utilizes these classes:

Officer | classes.py

Class that holds information about an officer within the code, rather than DB for quick access.

dispatchCall | kv.py

Class for building a call object and submitting it to the database.

report | kv.py

Widget class for popup screen for entering report information for a call

popupError | kv.py

Popup widget for error handling

DCADOfficerInfo | kv.py

Officer information widget within the officer box on the dispatcher screen

CallsBox | kv.py

A box widget on the officer screen that displays previous call information

OfficerBox | kv.py

A box widget on the dispatcher screen that displays currently online officers

OfficerScreen | kv.py

Box object that holds all other widgets for officer screen. Holds logic for when officer is logged in.

DispatchScreen | kv.py

Box object that holds all other widgets for dispatcher screen. Holds logic for when dispatcher is logged in.

LoginScreen | kv.py

Box object that holds all other widgets for the login screen. Holds logic and error handling for logging in, allows the transition between screens.

Other Classes:

Other classes were used, but had limited logic, mostly used for Kivy to run and create widgets. The ones above all contain logic, and functions to aid in functionality.

Project Modules

Modules are used for procedural based code that does not require state data like class modules do. Complete the introduction to modules.

`checkCall | callChecker.py`

Module that is run as a thread that checks and updates call information for the officer screen.

`checkState | callChecker.py`

Called within the `checkCall` thread, it checks the state of the officer from the database, and calls `flipState`.

`flipState | callChecker.py`

Gets the state of the officer, and updates the buttons on the officer screen accordingly.

`checkOnline | officerCheck.py`

Module that is run as a thread that checks for online officers, and updates the dispatcher screen according to the officer information such as online and status.

`checkOffline | officerCheck.py`

Called during `checkOnline` thread to check to see if an officer has gone offline, and remove them from the screen if they have.

`getCurCall | db.py`

Inputs an officer ID and returns their current call ID.

`setCallInactive | db.py`

Changes a call from active to inactive in the database.

`updateOnScene | db.py`

Changes an officer on scene state in the database.

`getCursor | db.py`

Gets a cursor object from a connection object for the database.

`getCNX | dbCred.py`

Builds and returns a connection object for the database and returns it

`build | tts.py`

Inputs a list of things to say, converts them into text to speech and plays the audio

`addNow | kv.py`

Adds the current time in the database for a call for on scene time

`getCallID | kv.py`

Gets and returns the current call id, allows for adding a call to the database with a unique id.

`updateAvailability | kv.py`

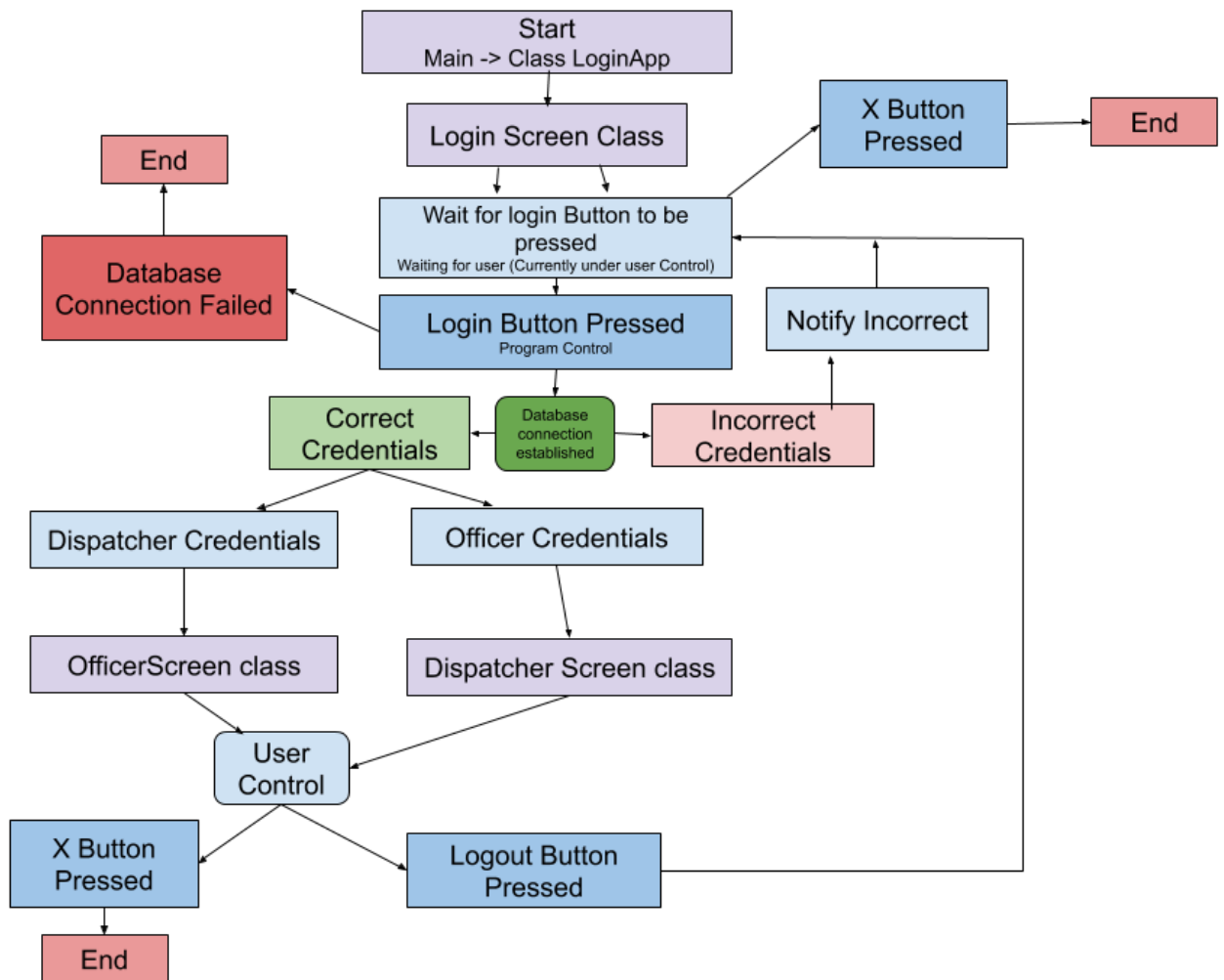
Gets the state of the officer, and updates the buttons on the officer screen accordingly.

`updateOnline | kv.py`

Changes the online status of an officer by their id using a Boolean, true for online.

Program Start and End Flow

This diagram outlines the main interactions for logging in and getting access to the screens to utilize the program. From building the application to granting the user full access to the program. First we start with logging in. Once the login button is pushed the connection to the database is now attempting to be established. Upon completion the program directs the user to the proper screen, then grants them full control.



Testing Plan

Testing is something that will need to be done upon each implementation of a new feature. One thing that testing needs to do is test each and every aspect of the program each time to make sure nothing is broken. One feature working on one small area of the program could break something elsewhere without knowing about it until testing. Testing periodically and often is an absolute must to catch bugs before they become too hard to find a solution to.

Testing will feature using multiple computers, and deliverables to make sure that everything is working as it should. Packaging a deliverable, running it in a script, running it on another computer, and running it via the IDE are all completely different. They all have different installations. Different scenarios, file paths, hardware, software, you name it. The conditions are different completely. Something may work in all but one of those settings but not on one. Testing will be conducted in various settings, various conditions and with various users to ensure that the system is working as it should be, and remains user friendly outside of the developer.

Deployment Plan

Deploying VCAD is the main focus of the project, as it must work in various environments across different platforms. It's intended to be used by a wide range of people and devices. It was built with this in mind, and Kivy is going to handle a lot of the brunt work of porting to different operating systems if the time comes for that. Testing is a major deployment process. Testing with various users and conditions is a must in the deployment plan.

There may be a process in which I allow a company to utilize VCAD for a while free of charge, to test on a commercial scale. Before the testing commercially is conducted, VCAD will go through a series of feature updates to prepare for commercial use for that specific company. Once VCAD is ready, it will be tested with several stress tests of many users utilizing the software at one given time. Once succeeded it can go to being used in the field.

Maintenance Plan

Once VCAD hits the field it will no doubt experience its series of bugs and down times. Watching the server hosting the database is going to be a major task, as it will likely be hosting VM's as well. If the system is down, the whole system is down.

If there are any bugs, they will need to be polished quickly as possible. When a bug is encountered, it should be submitted as a ticket so it can be fixed, and repackaged and updated to the client. As features roll in, the number of bugs will rise, and more time will need to be invested in maintaining bugs. This is when a new developer may enter the project.

Summary

The VCAD system is a great and simplistic way to help patrol officers for security companies to view call information on the go, and allowing dispatchers to monitor the company's officers in one simplistic place. This system will limit the amount of radio traffic that is being used, allowing for open waves for concentration and emergencies. VCAD will read out call information to the officer as the call comes in, keeping concentration on the road at all times, as well as utilizing a night mode design to aid in driving at night. VCAD aims to keep officers safe, and security companies efficient to aid their clients to the best of their abilities.

VCAD is built in a way that is optimized to grow and expand by using Python, a vastly popular programming language that has been growing immensely over the years. VCAD can be ported to all main operating systems, including mobile, by using Kivy, allowing room for expansion and ease of use for all officers. This system has more room to expand in its feature set, and can be easily modified for the specific needs from company to company. Fast deployment times can ensure that the system will always be up to par, and can fix bugs in a fast, and urgent matter. A database using MySQL has been utilized to store all of the information logged by the system, allowing for future access of the information for billing and insurance claims.

VCAD utilizes a lot of reusable code, allowing the code to run fast and efficient, as well as aid future developers to expand the system. Modules and classes have been developed with the future in mind. The technologies chosen have been implemented in a way that they can be changed and updated to newer iterations of these tools.

This document outlines all of the modules and classes that a developer would need to begin developing this project even further. A walkthrough for setting up client and developer machines can be found in the appendices. The database table sets can also be seen as a graphic within this documentation.

APPENDIX B (BUILD AND RELEASE PROCESS)

1. Commit to GitHub
2. Push to GitHub
3. In terminal CD to build directory
4. Add any dependencies to the top of VCAD.spec
5. Run command “py -m pyInstaller path/to/vcad.spec
6. Release the new build files within the directory the project was built to.

APPENDIX C (CLIENT INSTALLATION INSTRUCTIONS)

1. Proceed to GitHub at the link <https://github.com/KyleVenenga/VCAD>
2. Open the folder 'Build' and download VCAD-1.0-LH.zip
3. Open the folder 'Database File' and download the DB file
4. Import the DB file into a local MySQL database
5. Ensure that there is an account named 'vcad' with the password 'vcad123'
6. Start the server
7. Extract the zip
8. Navigate and run VCAD-1.0-LH/dist/VCAD/vcad.exe

APPENDIX D (DEVELOPER SETUP INSTRUCTIONS)

1. Proceed to GitHub at the link <https://github.com/KyleVenenga/VCAD>
2. Open the folder 'Database File' and download the DB file
3. Import the DB file into a local MySQL database
4. Ensure that there is an account named 'vcad' with the password 'vcad123'
5. Start the server
6. Clone/Download all of the python files, 'Audio' and 'Kivy Files' folders.
7. Ensure that all import dependencies are installed on local workstation
8. Dependencies can be found as imports in the .py files, and at the top of VCAD.spec (Note that vcad.spec does not include Kivy it is important to download kivy)
9. Dependencies for outside modules and libraries download varies based on programming environment, Google searching is the best option.