Smart Security Dash Camera (SSDC) Project

Matthew White, Scott Levine, Austin Sturm, Joseph Labauve, Timothy Deligero

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — This paper describes the functionality and purpose of the Smart Security Dash Camera (SSDC) project, which is a device that senses and tracks the immediate environment of a vehicle, and allows notifications to be sent to the user in the event of a burglary or accident involving the parked car in which the device is installed into. It utilizes both hardware and remote software in order to accomplish this task. This system is a more modern and safer car alarm, which allows the owner to take immediate action.**

***Index Terms* — Server API, GPS tracking, MEMS, Streaming Video**

I. Introduction

With the recent automotive break-ins on the UCF campus over the span of one night in spring 2017 it is apparent that there is a need for more in depth automotive security rather than just a horn alarm. The need for the 360 automotive security is evident. While this is a very recent noteworthy event there are many less known incidents that go unreported.

Fig. 1 Example of car damage and car theft caused to vehicle.

The project goal was to create an affordable product that can be bought and used as a vehicle theft deterrent device. This product is able to detect a disruption due to motion that is unwarranted to the vehicle. Once an anomaly is detected video will commence recording when an instance occurs. With the technology readily available to make this product the cost to produce this type of device for most young professionals and working individuals can be within affordable ranges, and become even cheaper as time passes.

II. System Components

The layout of the PCB for the SSDC is based on the arduino model 2560. The choice of the model 2560 as the base model was chosen due to many of the parts being of an acceptable accuracy for project specifications. The base model was modified for the needs of the project, and will control the sensory data as well as activating the Raspberry pi board. The Raspberry pi will control the cameras, and sending of the video. The main additions to the PCB compared to the base arduino model are the inclusion of a voltage regulator system to allow the device to be connected directly to a car battery as well as circuitry to support extra voltages as needed by different chips. Also a gps chip and cdma board were added with supporting circuitry.

A. Microcontroller

As stated in the introduction the PCB will be based on the arduino 2560. Originally a Ti board was considered for the base model of the SSDC. However due to the complexity of the board, along with the cost of manufacturing such a large multilayer board the choice was made to switch to the arduino 2560. The arduino will be essentially the brains of the project recording sensory data to check if an alert is raised. The ATMEGA 2560 chip is the MCU that controls operations of the PCB. Software will control the alerts the arduino sends to the raspberry pi through a rolling average to detect fluctuations in sensor data. Any alerts will cause the arduino to raise an alert pin to high, and the pin will be connected by a wire to the pi which will initiate the recording sequence. The data that the arduino measures are accelerometer, gps, and in the future magnetometer. To make better use of the space on the PCB board a number of GPIO pins were removed that were not required, and in their place the GPS chip was secured as well as the voltage system.

B. Raspberry Pi (mini-computer)

The Raspberry Pi 3 acts as the eyes that the SSDC PCB controls. It contains built in ethernet and a wifi chip allowing for mobile communication. The Pi 3 also contains 40 GPIO pins for communication; however a number of pins won't be used. Important features that are needed for the SSDC project are multiple usb ports for multiple camera usage, and a processor capable of handling HD video for streaming purposes. It is also important to note that the Pi comes with external ram to assist in the processing of the footage. While the base pi model has a majority of the features needed it lacked the gps, accelerometer, and a voltage regulator for jumping down a car battery's higher voltage. These areas however will be controlled and communicated by the SSDC PCB. When the PCB detects noticable changes in the sensor readings of the gps or accelerometer, and as a result the pins on the PCB will be raised to a value of high. These pins will be connected via a wire to the pi to signal to start recording.

C. MEMS Telemetry Sensor

For this project the LSM6DS3, an all in one chip, was used for the accelerometer and gyrometer data collection. This sensor was chosen due to its ability to detect within 2g to 16g motion, work in low power mode, and its ability to communicate through I2C or SPI. This proformance meets design requirements to detect motion, and to implement more advanced hardware for additional sensors that only provide minimal additional accuracy and precision would be considered over engineering.

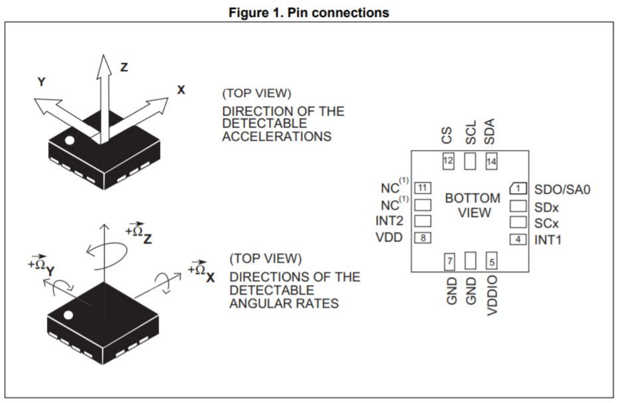
Figure 1. Pin connections
Z
Y
X
(TOP VIEW)
DIRECTION OF THE DETECTABLE ACCELERATIONS
(TOP VIEW)
DIRECTIONS OF THE DETECTABLE ANGULAR RATES
CS
SCL
SDA
12
14
NC(1)
NC(1)
INT2
VDD
11
8
BOTTOM VIEW
1
4
SDO/SA0
SDx
SCx
INT1
7
5
GND
GND
VDDIO

Fig. 2 This is the board view of the LSM6DS3 which contains the accelerometer and gyrometer used for detecting motion variants of the device.

D. GPS Module

The gps module chosen for the PCB was the FGPMMOPA6H. This was chosen due to its high precision of up to -165 db coupled with low power consumption. precision for certain components is important but it was a concern for gps, so the FGP module was chosen to detect any movement variations along with locating lost vehicles with high precision.

Fig. 3 The FGPMMOPA6H that tracks the GPS coordinates of the device.

III. Board Design

For the board design of the PCB some of the original board layout did not need to be tampered with however after removal of a number of GPIO pins, and the inclusion of key parts. Pieces had to be properly organized to prevent issues in frequencies as well as voltages. The GPS chip was isolated by itself on the right side of the board with the intention to reduce any noise that it may receive from digital components.

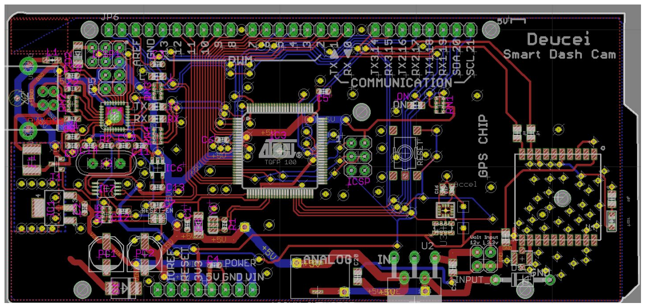

Fig. 4 PCB board design of the SSDC device.

Near the bottom of the board the voltage regulator was added with supporting circuitry obtained from its datasheet, and this placement was chosen due to the main voltage line from the base arduino board being located here. The new stepped down voltage will be fed in this main voltage line for simplicity, and additional smaller circuits were added to get the 3.3 volts for the accelerometer. The accelerometer module was added above the voltage regulator far enough away to guarantee proper clearance from any thicker traces.

IV. REST Server

An application programming interface (API) server established on a remote Amazon Web Services EC2 micro-instance acts as the communication medium between the mobile application and the hardware device. Without the central server the device would be unable to establish remote connections between the two devices due to a lack of address routing over unknown networks. The device and application both utilize Hyper Text Transfer Protocol (HTTP) requests in order to post data to the server. The protocol allowed for streamlined development of the communications over a common protocol. The downside to the protocol utilization being the increased packet size. The average HTTP request requires *"700-800 bytes"* [1], as opposed to the utilization of raw sockets which nearly only require the number of byte for the payload.

The Django Rest Framework, developed in Python3, handles the HTTP requests sent to the server. Utilization of a "View, Model, Serializer" design allows for the backend to function appropriately. The current implementation allows for large scale deployment, supporting thousands of Devices and Users. A user may register numerous devices to their account and manage them separately through the API.

Common security flaws: cross-site scripting, SQL injection, authentication bypass; All have been handled through secure programming practices along with built in security features to the Django Rest Framework. The framework provides automatic insecure input escape to protect against illegal characters through regex based string stripping function. Authentication implemented through the utilization of supported Token Based authentication allows for the users and devices to be only available to the owning account; Unauthenticated access through either invalid or missing token errors stating invalid access. Video feed security relies on an obfuscated url for each video stream, no authentication is built into this stream only the URL is required to view the stream. Future designs hope to implement an authenticated based streaming service.

The live video stream sends the data over the RTMP protocol to an NGINX relay hosted on the AWS micro-ec2 instance. The relay provides a public interface for the mobile application to receive the live video feed, without the relay the application would be incapable of reaching the video feed. The NGINX relay while rebroadcasting the stream as well saves the video locally on the device to allow the user to retrieve their stored streams separately from the local video on the device. Utilization of abstracted methods to achieve these goals decreased performance while allowing for an achievable development time.

V. Device Software

Device development utilizes core linux functionality along side development in both C and Python3.5. The device runs headless raspbian to support utilization of prebuilt linux utilities such as avconv/ffmpeg and wpa_cli to minimize development cycle. Avconv facilitates the streaming of the device through the Real Time Streaming Protocol (RTSP), over the provided connection medium (wireless, cellular).

The connection medium is determined through a custom wrapper over the linux, wpa_cli command line utility; The wrapper allows for automated network joining to either known secure networks as well as auto-join for open networks. In order to verify connection the device attempts to reach out to google's DNS servers (8.8.8.8), if no connection may be made an attempt to bypass any potential web-portal occurs. In order to bypass the captive portal a few challenges are presented, the main issue being the variety in captive portals as well as any potential captcha which may be presented on the page. Simple list of curl requests will be made to beat the majority of major captive portals, future implementation of a machine learning system to beat the captive portals will be implemented. A second attempt to verify connection occurs, upon failure the device repeats these steps for a new wireless network until success or complete failure.

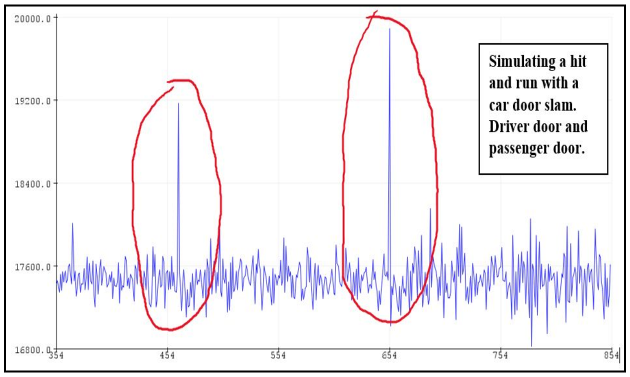

Fig. 5 Accelerometer testing for triggered events.

The statistical analysis of the device and decision making beyond the event queue rely on triggers and data sent from the secondary board to the primary. An interrupt listening on the primary board over a designated pin enables the camera's to record the event occuring. A circular buffer containing data collected at an equal time delta (Δt) computes the average of the statistical data to determine a baseline for anomalies. The tolerance must be high enough to avoid a false positive rate below 8%, with a 90% success rate.

The algorithm used utilizes a rolling average which stores a number of previous sensor input data and produces an average based on that dataset. The number of stored data has to be sufficiently large such that an accurate environment model is made, but also must be small enough so that a previously louder set of events does not affect the triggering logic in a more quiet time. This rolling average is used to compare to the current sensor values. If the difference is outside a threshold value, a trigger is sent to start recording and transmitting data.

Device decision making also occurs through the utilization of the mobile application. An event listener triggers periodically to obtain required actions, the event listeners blocks other actions as user events are prioritized over application events; except in the case of secondary board interrupt. The event listener class defines numerous available functions for the event queue listener to execute. The API issues commands in the format of: "method|arg1|arg2" which are parsed and executed by matching the method to the appropriate function available in the class and executed. Any event in the queue failing to match to an available function or failing to execute deletes from the queue ensuring only valid events remain. Upon successful completion the event is deleted from the queue.

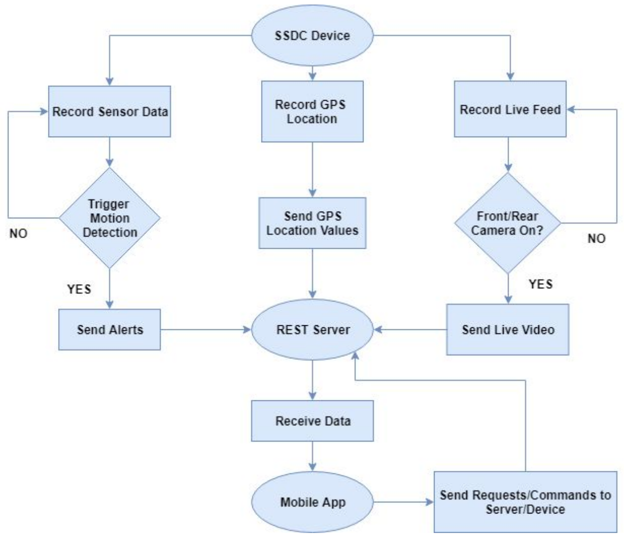

Fig. 6 SSDC, Server, and Mobile App interaction FlowChart

Logic allowing for utilization of the device over a software access point ensures availability within dead zones as well as initial setup. Interaction with the device over the local network requires separate functionality from that of the normal interaction. The interaction between the device and the mobile application must modify their communication channel as communication no longer occurs over the REST server. The device configures Layer 3 networking to static addressing with the device available at local IP Address: 192.168.0.100 and subsequent connected devices incrementally above ".100". Control of the device may be executed by sending specified data in the same manner used in the event queue management. A socket server listening on port 8888, listens for the specified format in the buffer and executes the called function.

VI. Mobile application

The mobile application is provided for users to interact with the server to send and receive data from the device for this project. The mobile application includes these features:

(1) Login/Register Accounts
(2) View/Receive Alerts
(3) View Live Feed from device's cameras
(4) View Vehicle Location with GPS
(5) Manage Stored Videos/Images
(6) Manage Devices/Networks
(7) Edit User Information

This section will provide a description on how the mobile application works and how its features were implemented.

A. Android Studio Software

The software used to develop the mobile application is Android Studio. Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA. This also includes features such as:

(1) A fast and feature-rich emulator.
(2) Instant run to push changes to the running app without building a new APK.
(3) Code templates (activities) and GitHub integration to help app developers build common app features and import sample code.
(4) Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine. [2]

The programming languages that are used to develop this app are Java and XML in the form of activities. The Java class files are used to respond to the user's inputs on the mobile application and send and receive data from the server and the XML files are used to format the data viewed by the user on the GUIs. Creating a new activity will create a code template with a Java file and XML file(s) to interact with. The mobile app will consist of many activities for the features mentioned above. The app will also utilize HTTP requests to send and receive data from the server within these activities when needed.

B. User Information

User information must be stored and used throughout the mobile app. This includes, the user's username, authorization key, the connected device's ID and primary key, etc. Android Studio provides an Application class that allows the mobile app to access global variable data from a single Java class with this extension. Activities will be able to access stored global data in this class without having to pass data through each activity.

The user has the option to change their personal user information for their username and email. Changing the user's username will require the user to input there current username and new username, while changing the user's email only requires a new, valid email address. After confirmation of these changes an HTTP GET request will receive the user's current user information from the server and add changes to the account to be saved to the user's account.

C. User Accounts

The user must either log into a previously created account or register a new account in order to access the main features of the mobile app. When logging in, the user must input valid user information, in the form of a username and a password, and after confirmation that the user has finished inputting their information, an HTTP POST request will be sent to the server with the inputted user information to validate that the account exists and is correct within the server's list of user accounts. If the user has not inputted information in either the username or password or if the server responds that the user information is invalid, then the activity will display a message to the user to input any missing user information or that their user information is invalid and to input their information again.

The user can also register a new account to the server, int the form of an email, username, and password. The new user information must be filled in, otherwise a message will be displayed to fill in the rest of the missing information. The user must input unique user information that does not match with the list of registered accounts in the server. The email must also be in a valid format and the password must be more than 8 characters long. After the user has confirmed creating a new account, an HTTP POST request will be sent with the new user information to the server to validate that the user information is unique.

When the user is logged into their account, they will gain access to an authorization key from the JSON Object received after the HTTP request if the user information inputted was valid. The authorization key is essential for many of the HTTP requests used in the features and activities in the mobile app. The authorization key itself will not be known by user and can only be obtain after logging into their account. This authorization key cannot be used when logged out of the user's account.

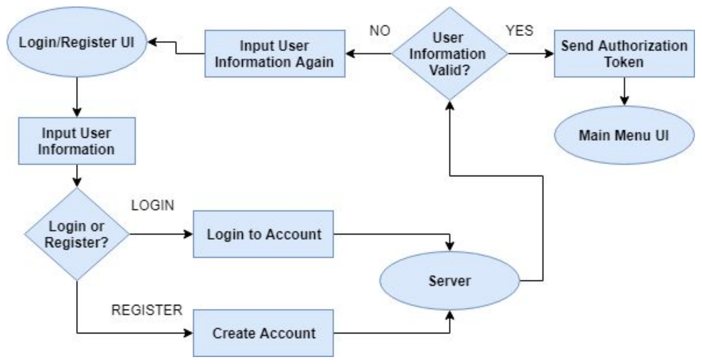

Fig. 7 User Login/Register functions Flowchart

D. Devices and Networks

In order to interact with the device, or have the ability to receive alerts, view the device's location or live feed, and view any stored media, the user must first connect to a device and gain access to their ID and primary key, which is essential for HTTP requests used in other activities to receive data from the server. The user must add devices to their user account via inputting the device's ID and using an HTTP POST request to add that device to the user's account, which will be displayed in a list with multiple options on each device. The list of devices will be obtained using an HTTP GET request to the server to receive the all devices stored in the user's account and each device's information.

If there are devices available to the user in their list, they have the option to clear alerts, connect to the device, delete the device from their list, or view the network list for each device. Clearing the alerts will send an HTTP DELETE request to clear the alert queue in the device. Connecting to the device will store the device's ID and primary key to be used in other features of the app. Deleting a device will send an HTTP DELETE request to delete the device from the user's list of devices. Viewing the networks will send an HTTP POST request to get the device's networks based on its ID and will list the networks that the device is connected to.

The user will have the option to add networks to the device to connect to. The user must input the SSID, PSK, and the network type, either WEP, WPA, WPA2, or None as selective options. After the user confirms to add this network to the device, then an HTTP POST request with the network information will be used to add the network to device for the user to view in the networks list. This will include the SSID, PSK, and network type information for each network.

E. Alerts

The user will be able to view a list of alerts recorded by each device. This will include information of the type of alert, time recorded, the details of the alert, and which device the alert originated from. These alerts will be obtained using an HTTP GET request to receive alerts from each device the user has added to their list of devices. On default, if no device is connected, then the user will view all alerts listed from every device that is listed in the server. If the user is currently connected to a device, then alerts will only be listed for that device. If there are no alerts currently listed, then it is displayed that no recorded alerts are provided for that device.

The mobile app will detect for new alerts from a connected device. This will be done using a Timer function provided by Android Studio that executes functions periodically given a period of time. Between each execution of the Timer, an HTTP GET request will be sent to look at the most recent alert of the alert queue. The most recent alert will be analyzed of the time it was recorded to ensure that this alert has occurred at recently from the current time. If the code detects that the alert is recent, then a push notification will be sent to the user's mobile device to notify them that car theft or car damage has occurred to their vehicle. The most recent alert will also be recorded to ensure that only one push notification has been sent for a new alert. These push notifications will be done in the background of the mobile application by utilizing the Service component of the Android Studio. A Service is an application component that can perform long-running operations in the background, and it does not provide a user interface. Another application component can start a service, and it continues to run in the background even if the user switches to another application. [3] This means that a user can access the features of the mobile app and be able to use different apps while the mobile app will continuously detect for new alerts. The Service and Timer are cancelled once the user logs out of their account with confirmation that they will no longer receive anymore push notifications/alerts from the mobile app.

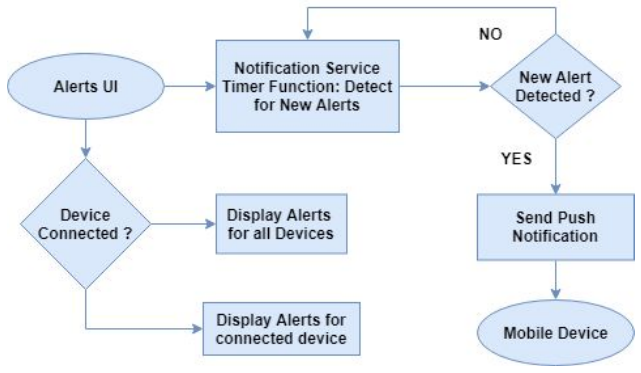
Alerts UI
Notification Service Timer Function: Detect for New Alerts
NO
New Alert Detected ?
YES
Device Connected ?
Display Alerts for all Devices
Send Push Notification
Display Alerts for connected device
Mobile Device

Fig. 8 Alerts functions Flowchart

F. Vehicle Location

The mobile app allows the user to view the device's/vehicle's location using the GPS data from the server received from the device. The mobile app uses the Google Maps API and the Google Maps Activity template to display the location of the user's vehicle. A device must be connected in order to obtain the specific GPS data of the device. Once the user is connected to a device, a map will be displayed showing marker of the user's vehicle location. The GPS data will be received using an HTTP POST request to receive the GPS data in the form of latitude and longitude values for the connect device from the server.

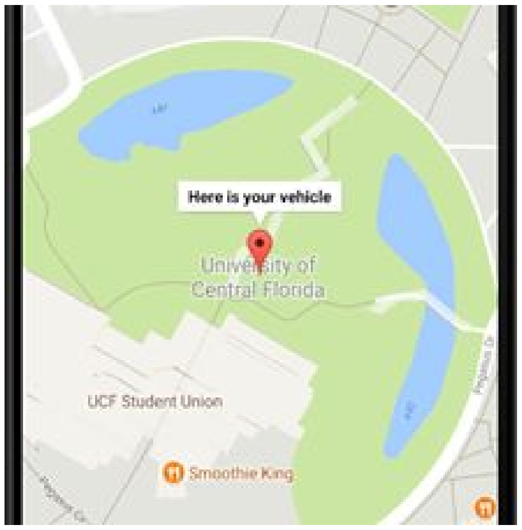

Fig. 9 Example of the GPS location of the vehicle in the mobile app.

In order to update the vehicle's position, a Timer function is used to execute the POST request at a certain time period to receive the updated latitude and longitude values recorded. This is also used to locate the vehicle in case of car theft or if the vehicle potentially moves in some other cases. The marker will move and the map will focus on the vehicle's location on every update.

G. Live Feed

The user will be able to view live feed from the front and rear view cameras of the device through the mobile app with two interfaces for each view. Android Studio provides VideoView widgets that allows the mobile app to display videos with a URL connection string provided for the video. The URL would be parsed and converted into a URI object reference, which is used to identify and reference a resource, in this case a video. By setting a Video URI reference to the VideoView, the user will be able to view the live video in current time. The live feed will be received using an HTTP GET request to take the URL string of the live feed from the connected device. There will also be boolean values received from the request as well to indicate if the cameras are currently on or off. The user will have the options to toggle the cameras ON or OFF or switch between the front view or rear view cameras in an options menu in the interface.

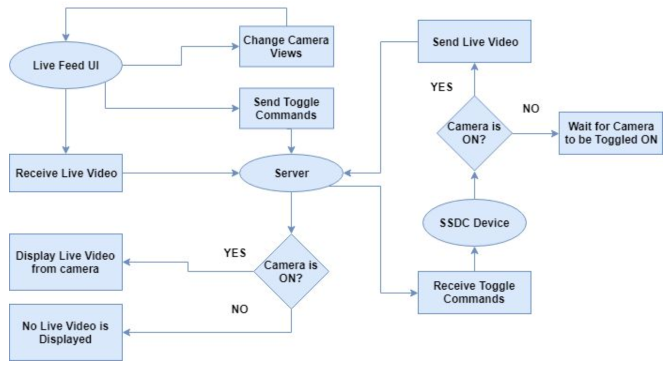

Fig. 10 Live Feed flowchart of toggle commands and displaying live video.

Unfortunately, Android Studio, as of version 3.1, supports limited network protocols for audio and video playback.. This makes some URL connection strings unable to use and view the video from the mobile app. The live feed from the device is using an RTMP protocol, which is not supported in the current version of Android Studio. Instead, Vitamio, an open source library, is used in order to compensate for the Android Studio's limited uses with VideoView. Vitamio is an open multimedia framework or library for Android and iOS, with full and real hardware accelerated decoder and renderer. Vitamio can play 720p/1080p HD mp4, mkv ,m4v, mov, flv, avi, rmvb, rm, ts, tp and many other video formats in Android and iOS. Almost all popular streaming protocols are supported by Vitamio, including HLS (m3u8), MMS, RTSP, RTMP, and HTTP. [4] Vitamio has its own custom VideoView that works similarly to the VideoView widget in Android Studio, but is able to accept many network protocols and media formats. With this included to the mobile app, live feed can be received and viewed by the user.

H. Stored Media

The device will send recorded images during instances of recorded alerts to the server. The user will be able to view the recorded media stored in the device. The user must first connect to a device in order to receive the stored media. The user will receive the stored media by using an HTTP GET request to receive any stored images for the device based on the connected device's ID.

The images will give the user the options to view the full recorded image or delete the image from the server. The list of images will contain Bitmap values from decoded streams from the stored image's URL path. Android Studio provides an ImageView widget that allows images to be viewed by the user on a mobile app. The Bitmap values will be used to set the ImageView widgets so the user can view the stored image. When deleting a stored image from the server and device, the URL path must be given to remove the reference and data stored. This will send an HTTP DELETE request to the server to delete the specified image.

I. Testing

Android Studio provides an emulator to test the mobile app on an emulated Android device. When the mobile app is being run, the processes are recorded in the Logcat of Android Studio, which can also be used to debug the mobile app code. Whenever the app crashes or has errors, Fatal Exception errors are recorded in the app to detect the reasoning and where in the code that causes the app to crash. This is used to improve the mobile app by fixing these issues. The following features for this project's app are tested multiple times on the emulator to detect any errors or potential issues within the code.

However, the emulator from Android Studio is not always reliable, as it can be comparably slow to actual hardware devices and there have been instances of the emulators being buggy when run on certain emulated devices. The mobile app code can run on an Android device by having a built APK. The APK can be emailed and downloaded onto an Android device to be run and tested. The mobile app on the Android device will work the same as the emulator on Android Studio.

VII. Conclusion

This project has provided valuable experience to the whole group on learning how to work and plan in developing an actual product. Throughout the group's experience, there was an emphasis that contribution and effort in ideas, planning, and giving different roles and responsibilities to each member are important in developing a project. Despite having difficulties, issues, and learning curves during the process, it became valuable experience to learn the hardships and obstacles it takes to creating a product. Overall, the group has learned and developed their skills as engineers that will be valuable in their future professions.

The engineers

Matthew White, senior Electrical Engineering student at The University of Central Florida College of Engineering and Computer Science, will start to pursue his career as an entrepreneur. He has accepted an offer to work for a large fortune 500 company. He will continue working on his own business. He joined the Upstarts program at University of Central Florida. During his time at UCF he has placed 2nd in the Business Model Competition and 2nd in a statewide business competition. He has spent more time in the business building then the engineering building at UCF. He owns a US Patent and plans to have many more under his name in the upcoming years.

Austin Sturm, senior Computer Engineering student in the College of Engineering and Computer Science at the University of Central Florida. Employed at the Walt Disney World Company for the past two years as a Security Analyst. An avid member of Hack@UCF, competing in numerous security competitions across the nation.

Timothy Deligero, a senior Computer Engineering student at University of Central Florida graduating after spending 3 years in the university. After graduating, he will be enrolled in to a graduate school in UCF to earn a Master's Degree in Computer Engineering and pursue a job in the same background.

Scott Levine, senior Electrical Engineering student in the College of Engineering and Computer Science at the University of Central Florida. Hopes to pursue a career in the automotive field.

Joseph Labauve, senior Electrical Engineering student in the College of Engineering and Computer Science at the University of Central Florida.

Acknowledgement

We would like to acknowledge Samuel Richie for continued support and coming out of retirement to be our Senior Design teacher at UCF. Jerry Reed is a Valencia Professor that has helped immensely on our project coding.

References

[1] "SPDY: An experimental protocol for a faster web" *The Chromium Projects*, http://dev.chromium.org/spdy/spdy-whitepaper.
[2] "Meet Android Studio." *Android Developers*, 26 Mar. 2018, developer.android.com/studio/intro/index.html.
[3] "Services." *Android Developers*, 13 Mar. 2018, developer.android.com/guide/components/services.html.
[4] "What's Vitamio?" *What's Vitamio? - Basic Introduction*, www.vitamio.org/en/docs/Basic/2013/0509/4.html