Sparrow

The Dashcam Defender

Final Report

Dec20-11
Evan Timmons, Cobi Mom, Danny Yip,
Scott Vlasic, Ismael Duran, Durga Darba

1. Table of Contents

2. Glossary

Term	Definition
ALPR	Automatic License Plate Reader | A system that automatically detects and records the numbers and symbols of a license plate
AVI	Audio Video Interleave | Known by its initials AVI and the .avi filename extension is a multimedia container format introduced by Microsoft
GUI	Graphical User Interface | A form of user interface that allows the user to interact with software through icons rather than solely through the command-line interface.
JSON	JavaScript Object Notation | An open standard file format and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute-value pairs and array data types.

3. Executive Summary

Development Standards & Practices Used

Software practices

- Make code correct first and fast second
- Always test your code
- Agile Methodology
- Integration with hardware
- Simple design
- Pair programming
- Continuous integration

Hardware Practices

- Integration with software
- Continuous validation testing
- Agile Methodology

Summary of Requirements

- Hardware development
- Mobile Application Development
- Database Design

Applicable Courses from Iowa State University Curriculum

- ComSci 309
- CPRE 288
- COMS 228
- SE/CPRE 185
- COMS 363
- CPRE 310

New Skills/Knowledge acquired that was not taught in courses

- Hardware Assembly
- Dart Programming Language
- Flutter Framework
- Hardware / Software integration
- API Calls
- Angular

4. Project Design

○ General Overview

Approximately 2 million people experience severe injuries from car accidents each year in the United States. We hope to drastically reduce that number with Sparrow, a product designed to automatically scan license plates and alert users to reckless drivers.

Sparrow is a smart dashcam system that is intended to increase the safety of those who use it. The system is composed of a few main components, a dash camera, an ALPR board, a smartphone app, and a backend database.

The ALPR board takes video input from the dash camera to read and interpret license plate data of the cars around the user in real-time. Then the board uploads the recorded video and JSON file of the detected plates to the database. The mobile application is then used to browse the database video files by the license plate identifiers.

In a final product, these systems will be used in tandem to alert drivers when a hazardous driver is near them. If the plate of a driver that has been properly flagged as dangerous is detected by the user's system, then both an audio and visual alert will prompt the user on their smartphone.

The system aims to make a strong impact beyond just the safety of its immediate users. Through careful restricted access protocols, law enforcement could use the system to find stolen vehicles, fugitives and even amber alert vehicles with locational data. As a police officer that we interviewed eloquently put it, this system will "be a ring doorbell for the road".

○ Problem and Project Statement

The roads as we know it are full of reckless drivers. What if there was a way to automatically detect when you are near a reckless driver? The solution that we are proposing is the ability for everyday consumers to come together and crowdsource information on reckless drivers. This is done by utilizing Sparrow, a product designed by us to automatically scan license plates and report reckless drivers when they are encountered instantly. Reporting is supported by a public facing website and app. This way, users can look up and report reckless drivers from multiple fronts.

○ Operational Environment

Since the product will be mounted on the windshield, it needs to be able to handle the fluctuation in temperatures that may occur in the car. Therefore, we will have to construct a container to protect the device from severe weather when the car is unattended as well as from consumer damages. This container must also be able to handle rough road conditions and keep the camera stable. Another condition we must consider is the camera's ability to capture data given poor weather. To ensure this, we will have to use a camera that returns images in 1080p or higher.

○ System Prototype Layout

Figure 4.1: Featured: Mobile Phone, ALPR Board and Dash Camera

○ Product Diagram

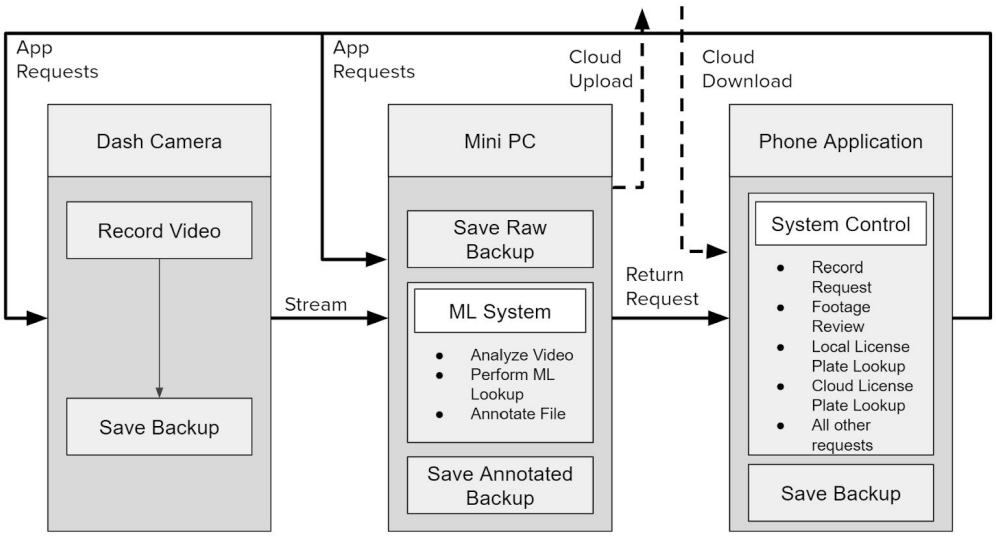


Figure 4.2: General Product Diagram to Show System Interaction

- ## Engineering Constraints and Non-functional Requirements
  - ### Constraints
    1. The final product consumer must not cost more than $1,000
    2. The system must comply with all applicable safety and legal regulations governing commercial automobile electronics.
    3. The product must be powered through a standard 12V car outlet.
    4. The amount of time to develop is not too long, because we only have 2 semesters to work on it.
    5. It is hard to collaborate with the team due to COVID, and it makes it harder for the hardware team to fix any technical issues.
  - ### Non-Functional Requirements
    1. The mobile application and GUI shall respond to each button press within .5 seconds of release
    2. The mobile application should make sure the user's password was encrypted while storing into the database.
    3. The system shall be able to record video at 60 frames per second.
    4. The system shall be able to analyze ALRP video at 60 frames per second.
    5. The system shall process and store videos of 1080p resolution.
    6. A typical new user shall be able to fully understand and operate the system within five minutes of initial contact.
    7. The system shall be able to accurately read license plates at a speed differential of up to 60 miles per hour.

5. Implementation

- ## Mobile Application Implementation

  We have chosen Flutter for our mobile application implementation. The main reason is because we would like to learn more about flutter. Flutter can be used to develop a mobile application fast, flexible, and provide a native performance experience to IOS, Android or web applications. Flutter is a fast development framework, and because of its hot reload feature, we can shorten the time to build UI, add and remove features, and fix bugs. In addition, thanks to Material.io, we can build beautiful apps in Flutter with the minimum time, and at the same time, the controls and usage are user friendly. Last but not least, flutter is a good platform to develop in multiple platforms at the same time. Therefore, by using flutter, it will be easy for us to maintain the code, especially when more codes are implemented in the future.

- ## ALPR Board Software Implementation

  For the software running locally on the ALPR board we focused on functionality over all else. The board needed to have an interface allowing drivers to start and stop recordings with ease. Thus, we need to implement a recording program, a ALPR program and a script to upload the files to the database all controlled by a custom GUI. The GUI needed to be simple to use yet still tie together all of the aforementioned functionality.

After researching and testing multiple options, we settled on the python based PySimpleGUI package to aid in our GUI development. The package allowed for fast paced prototyping, cross platform support and high system compatibility. The GUI shows a real time preview of the camera yet needs to record the stream and send it to ALPR software at the same time.

This proved challenging given that the board was running linux, an Operating System that does not inherently support using one camera stream for multiple purposes. Eventually we were able to produce a solution that worked by creating virtual cameras that duplicated and streamed data from the original dash camera. When the user presses record in the GUI the system then prompts the recording software to record an AVI video. Additionally, the ALPR will initiate and output the license plate data in the form of a JSON file.

The final part of the ALPR Board's software implementation is the code that works to upload the AVI and JSON files to the database. The script creates names for both the video and JSON files with a unique identifier and then uploads them to the S3 bucket and database respectively. Then, further processing of the files happens on the AWS system.

○ Cloud Implementation

Mobile Application Database API

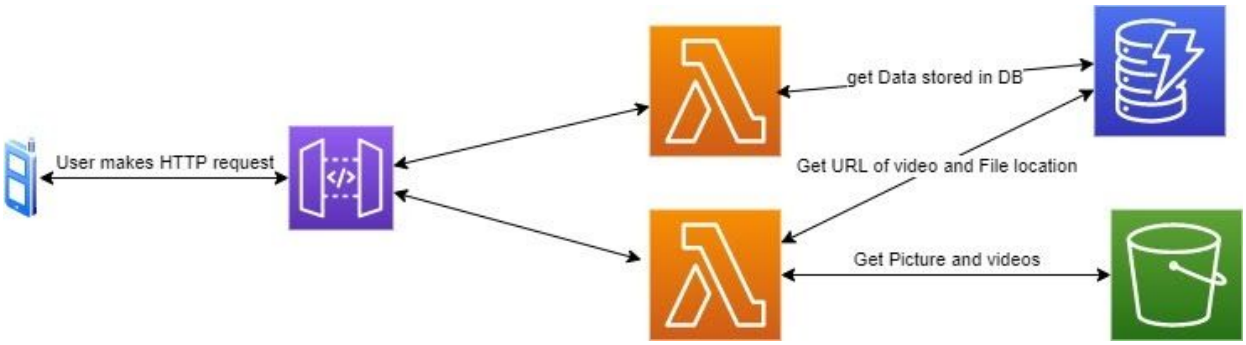


Figure 5.1: Mobile Application Database API

API Gateway
The gateway serves as a singular endpoint for our HTTP requests. With API Gateway we can create an HTTP request that can easily be connected to a Lambda function.

Lambda Function
The lambda function is the computation needed for requests. It is triggered by the API Gateway when called upon. The function is able to interact with the database in many ways, including read, write, modify, add, etc.

DynamoDB
DynamoDB is a NoSQL database. Our purpose for it is to store user information and video location file paths that will lead to videos on the S3 bucket. It also houses license plate information gathered by users.

S3

S3 is a file system storage that Sparrow uses to store its videos. In relation to the mobile application, it will get video URLs given by AWS to stream via the mobile application.

○ Hardware Implementation

We have chosen the NVIDIA Jetson TX2 as our hardware device to run the embedded software and OpenALPR. We chose the Jetson because it is more powerful than other hardware devices like the Raspberry Pi, which was necessary to run machine learning algorithms. In addition to its superior hardware metrics, the Jetson TX2 also had documented use with the OpenALPR software.

In addition to the board itself, there were many peripherals needed for the hardware implementation. A third party camera connected to the Jetson TX2 to film and record footage of the user driving with the OpenALPR software running. To add storage to the device, a 256 GB SD Card was mounted on the board since video files take up a large amount of space. Another peripheral connected to the board was a small LCD display to display the recording to the user while they are driving.

6. Testing

○ Mobile Application Testing

Since we are using flutter, to test the mobile application, we use the flutter_test dependencies package to do some testing. We have implemented a few tests, for example, we test for the login page to make sure that users are able to successfully login if only they have email and password that matches the database.

Besides, we also did some tests on the other class to make sure that it is working correctly. In addition, we also test it manually, by using the Android phone emulator. We run the application on Android phone emulator, and click on all the functionality that we have. To make sure it runs correctly, we try to run the application again and again, and make sure that it is working like what we expected.

○ Database Testing

To test the database with the board, we began by creating a prototype program on our PCs that sends pre recorded videos to the AWS S3. From here, we moved the upload script to the Nvidia Jetson and began looking at how license plate data saves and sends the video back to the bucket.

Furthermore, to ensure the video is associated with different users, we sent a JSON object to the database with a path to each video file for each user. Similar to testing the videos, we ran this script from our laptops to ensure the data was being processed correctly. From here, we moved the script to the board to ensure the data can be processed and sent correctly.

Finally, we ran the upload script and the json script in tandem such that when the upload script ensures the video has been uploaded, it updates the database tables with the correct user information.

○ ALPR Board Software Software Testing

The Sparrow GUI was designed and implemented in Python allowing for straightforward testing. The GUI was tested with verification, validation and uncertainty testing. For verification testing, each iteration of the GUI (v1, v2 etc) was validated by multiple users to be bug free and repeatable in normal operation.

Validation testing was done through the use of built in Python functionality. Using custom methods and asserts, we were able to test and assure that the program was operating as we expected it to. For instance, a test was used to check the output directory and ensure that an output video and JSON file were being stored upon completion of the video recording process.

Finally, uncertainty testing was completed by simulating edge and corner cases in our code. The GUI allows for direct interaction with the end user, so we needed to ensure that the program would work with all combinations of user input (intended and unintended). Additionally, tests were conducted to verify continued operation in the case of file errors. If the software records a video and tries to save it to the system with an existing file identically named, the system should continue to operate. Similarity, if the video recording or ALPR process fails, the system continues to run without crashing.

○ ALPR System Testing

To test our OpenALPR system, we first began by ensuring that the software was able to recognize our third party camera and read license plate images when uploaded to the device. Once that was confirmed, we transitioned to feeding the software a video of different license plates and recording the accuracy of the detection to certify that the detection rate of the software was viable for our project scope. Our viability threshold was a license plate detection of 80% or higher and the open source software was able to detect the correct information 83% of the time.

Once the threshold was confirmed, we determined a target rate of detection per minute that we would like to achieve, plates per minute. Given a video stream and taking the accuracy measurements into account, we wanted the average user to be able to detect up to 5 correct plates per minute in a high traffic situation. Using the GUI, we took note of how many license plates were detected per minute from a high traffic videostream and averaged it over the total duration of the video. Our findings were that the software detected ~5.7 plates per minute, above the target rate we had set.

7. Project Context

When doing research on license plate aggregation tools early on the project, we noticed more and more instances of products that do very similar things. Rekor Systems is a license plate detection tool that's used as a way to monitor traffic and send reports of vehicles. When building Sparrow, we used this tool as inspiration. OpenALPR is an open source software we discovered early on that helps us get license plate numbers off images of license plates and associate them with the state the car is from. Altogether, both of these products helped us create the functionality behind Sparrow.

The idea of recording a user's license plate information can manifest concerns about privacy. These concerns are shared among us as members of the development team as well, and we take them incredibly seriously. A large governing reason for the continuing development was best stated by

someone we interviewed for feedback on the project: “If [we] do not create the idea, then someone else eventually will, and they may have worse intentions”. So, given that we believe the Sparrow system has strong potential for good in the world we choose to continue with careful and deliberate development.

As part of this cautious development, we have amended the original concept multiple times. One such amendment proposes the use of a transparent system for law enforcement to request data access. For instance, if law enforcement were to demand data about a vehicle claiming that it had been stolen the company would send a letter to the registered owner of the vehicle to ensure transparency for all parties. The owner of the vehicle will have to approve the request before law enforcement continues. Such ideas will continue to be adapted into the growing Sparrow concept. Additional information about law enforcement using the Sparrow system is contained in Appendix III.

8. Appendix I – “Operation Manual”

- ## ALPR Board Operation
  - ### Initial Setup
    1. 7 Inch Display
       a. Plug in the associated power supply to the provided 7 inch display
       b. Plug the other end into your power source
          i. A standard 120v wall outlet
          ii. Or, the provided inverter, plugged into an automobile outlet
       c. Attach one end of the provided HDMI cable into the display
    2. Jetson Board
       a. If not already attached, screw in provided WiFi antennas to the ports on the side of the device
       b. Plug in the power supply to the NVIDIA Jetson TX2
       c. Plug the other end into your power souce:
          i. A standard 120v wall outlet
          ii. Or, the provided inverter, plugged into an automobile outlet
       d. Connect the keyboard/mouse receiver into the USB-A->MicroUSB adapter
       e. Plug the adapter from step 3 into the MicroUSB port on the board
       f. Turn on the keyboard using the switch on it’s top edge
       g. Insert the other end of the HDMI cord (from the display, mentioned in step 1c)
       h. Plug the web camera into the USB-A port on the device
  - ### ALPR Board
    1. Turn on the NVIDIA Jetson TX2 by holding holding the power button until the lights turn on
    2. Login with the following credentials
       a. Admin Username: Senior Design
       b. Password: wallace103
    3. Open a new instance of terminal
       a. Run the custom command to create a new virtual camera
          i. `sudo newVCam`

4. Open another new terminal instance
    a. Run the custom command to stream the web camera input to the virtual camera
        i. `videoSplit2`
5. Navigate to the SparrowGUI directory
    a. cd home/senior-design/SparrowMaster/GUI
    b. Open a new terminal instance in this location
        i. Run the following command to start the GUI

```
1. python3 SparrowV2
```

SparrowV2 Graphical User Interface

1. Press the record button, which will:
    a. Initiate video recording capture

Figure 8.1: GUI process to begin video capture

    b. Initiate ALPR analysis of the live footage
2. Press the stop button to:
    a. Stop video recording and save an AVI output

Figure 8.2: GUI process to stop recording

b. Stop ALPR analysis and save a JSON output

3. Press the upload button to:
   a. Upload the most recent AVI and JSON outputs from the program

Figure 8.3: GUI process to upload to Amazon S3 bucket

○ Mobile Application Operation

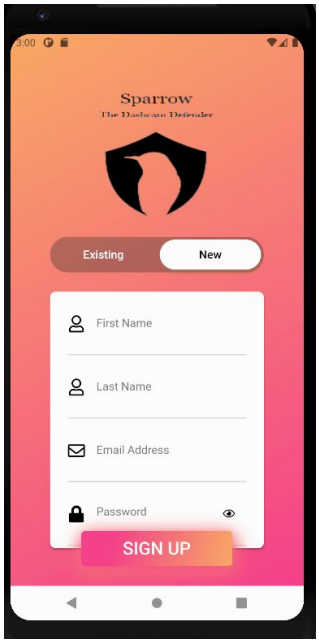


Figure 8.4 : Signup (Step 1)

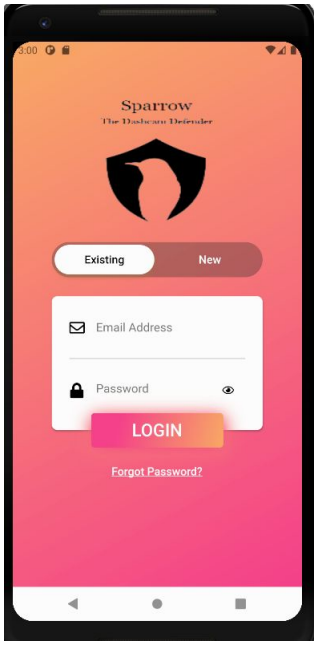


Figure 8.5 : Login with email and password (Step 2)

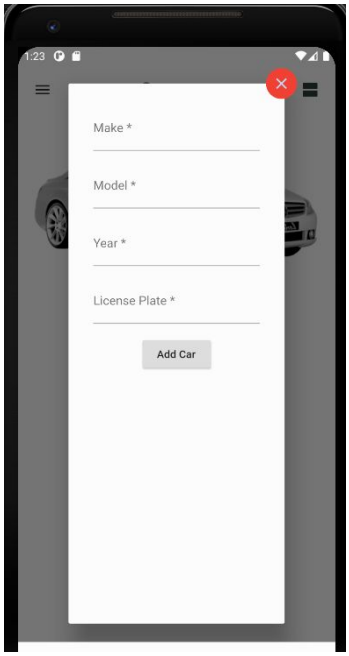


Figure 8.6: Add a car (Step 3)

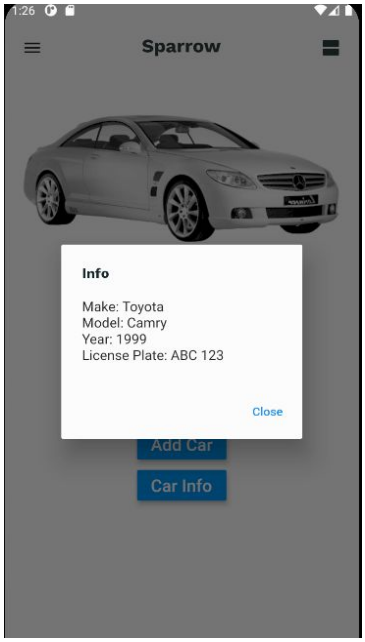


Figure 8.7: Press info to see details (Step 4)

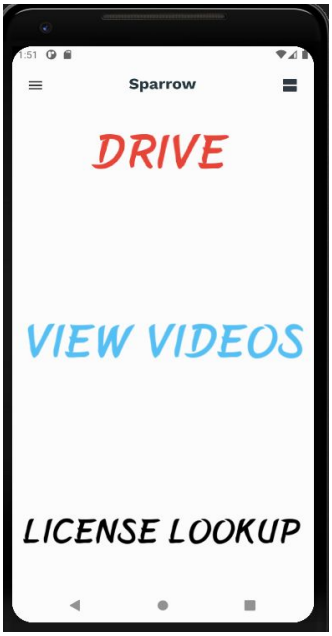


Figure 8.8 :Press the car picture (Step 5)

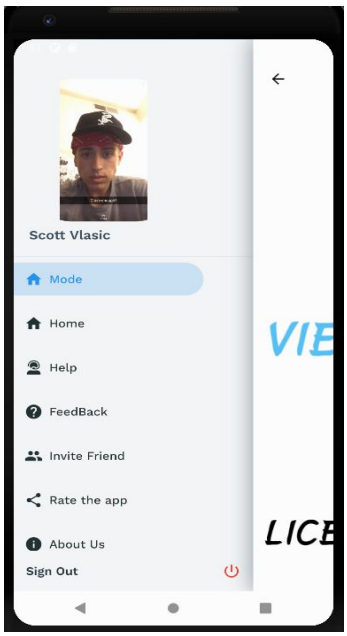


Figure 8.9 :Press top right button (Step 6): Navigate screen & sign out

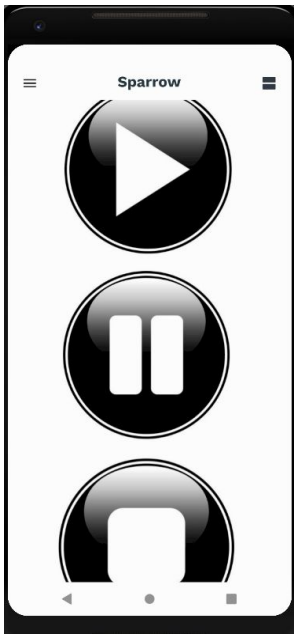


Figure 8.10: Press Drive button (Step 7): start record & stop

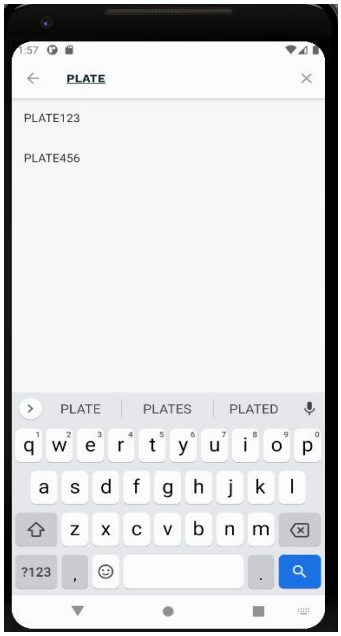


Figure 8.11 :Press License lookup (Step 8): search for car plate

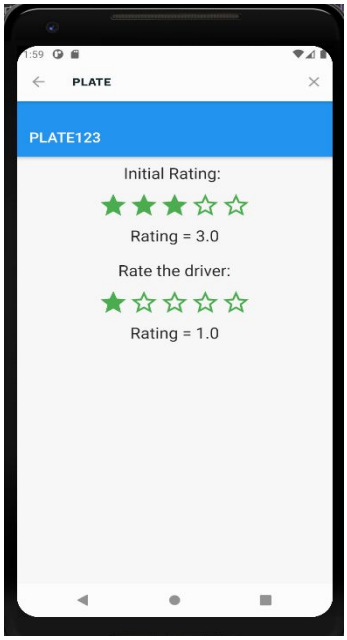


Figure 8.12 :Select a plate (Step 9): See rating, and able to rate

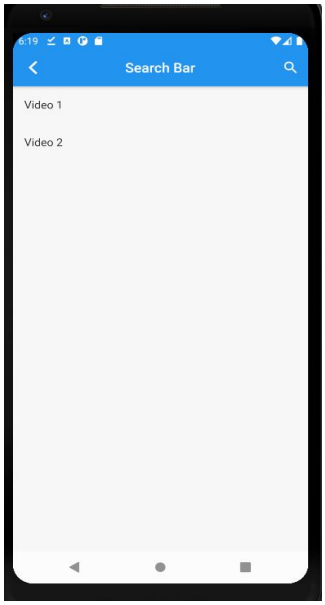


Figure 8.13: Press VIEW VIDEOS (Step 10): search videos

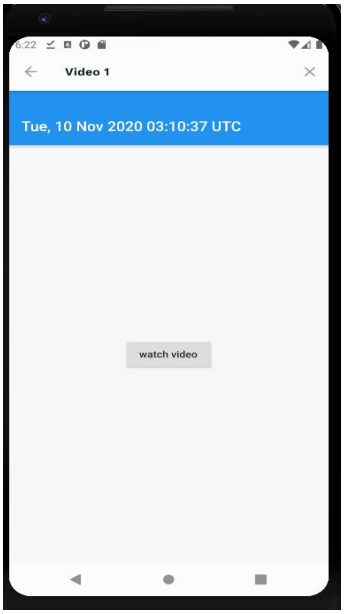


Figure 8.14 :Search a Video (Step 11): display the timestamp

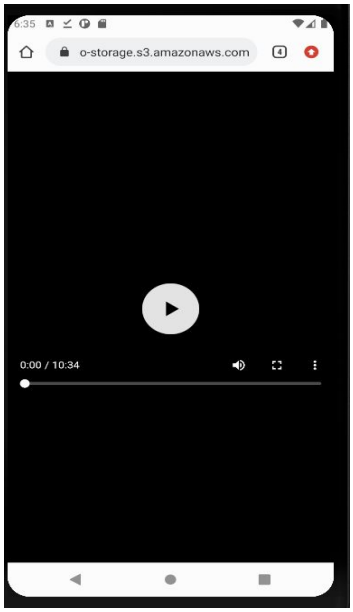


Figure 8.15 :Select to watch video (Step 12): videos pull from S3 bucket

9. Appendix II: Alternative Designs

○ Web Application

When the COVID pandemic began last semester our team had to make some large changes to our project plans. Given the circumstances, and various locations of team members, we needed to pause development of the ALPR board itself. Instead, part of our team transitioned to developing a web application for the Sparrow board.

We spent time investing different web app platforms and writing prototype software for the application. The intended use case of the webapp was to browse and rate videos as well as manage user account settings all from a hosted website (prototype image posted below). Once we returned to campus this semester (Fall 2020) we decided against further development of the web application. In order to make the most impactful progress on the project, our team thought it best to return to the development of the hardware itself.

The decision proved to be a wise one, as we were able to create a working prototype by the end of the semester. The mobile application team took on some of the features that were initially intended to be part of the web application so we retained functionality.

○ ALPR cloud computation

One potential improvement that we can make for the dashcam defender is getting rid of the Jeston board and have the computation done on the cloud. Since the late decision of using AWS as a team, we realized that the ALPR could be run on the cloud. It will eliminate the use of the jetson board for computation. The idea has the camera getting the video uploaded or streamed to a cloud service. It will reduce the physical equipment that we currently have, which will correlate to the product's lower cost. The Jetson Nano is costly that will make it more affordable for the potential of more users.

- **Design Document from CPRE/EE/SE 491 (Pages 18 - 41)**

Executive Summary

Development Standards & Practices Used

Software practices

- Make code correct first and fast second
- Always test your code
- Agile Methodology
- Integration with hardware
- Simple design
- Pair programming
- Continuous integration

Hardware practices

- Integration with software
- Always test and make sure it is working
- Agile Methodology

Summary of Requirements

- Knowledge in mobile development
- Hardware development
- Database Design

Applicable Courses from Iowa State University Curriculum

- ComSci 309
- CPRE 288
- COMS 228
- SE/CPRE 185
- COMS 363
- CPRE 310

New Skills/Knowledge acquired that was not taught in courses

- Hardware Assembly
- Dart Programming Language
- Flutter Framework
- Hardware / Software integration
- API Calls
- Angular

1. Introduction

1.1. Acknowledgement

We thank Dr. Joseph Zambreno for his consultation as our senior design advisor. We acknowledge the development done by our team members Evan Timmons, Cobi Mom, Danny Yip, Scott Vlasic, Durga Darba, and Ismael Duran.

1.2. Problem and Project Statement

The roads as we know it are full of reckless drivers. What if there was a way to automatically detect when you are near a reckless driver? The solution that we are proposing is the ability for everyday consumers to come together and crowdsource information on reckless drivers. This is done by utilizing DashCam Defender, a product designed by us to automatically scan license plates and report reckless drivers when they are encountered instantly. Reporting is supported by a public facing website and app. This way, users can look up and report reckless drivers from multiple fronts.

1.3. Operational Environment

Since the product will be mounted on the windshield, it needs to be able to handle the fluctuation in temperatures that may occur in the car. Therefore, we will have to construct a container to protect the device from severe weather when the car is unattended as well as from consumer damages. This container must also be able to handle rough road conditions and keep the camera stable. Another condition we must consider is the camera's ability to capture data given poor weather. To ensure this, we will have to use a camera that returns images in 1080p or higher.

1.4. Requirements

Project requirements

- Dash camera that can record in 1080p
- A powerful enough computing board that can process the machine learning application
- A computation board that can work with peripherals

Functional Requirements

- A license plate reader that can accurately read license plates from nearby cars
- A platform where the user can report and view information

UI requirements

- Have an aesthetically pleasing but simplistic application for the user
- Have a driver mode focused on simplicity for maximum road attentiveness

1.5. Intended Users And Uses

We have three major intended users for our product:

- Police Departments
    - Police can lookup license plates and their last known locations to solve crimes faster
- Everyday Drivers

    - The everyday driver can report poor drivers and get alerts to notify them of when they are in the presence of a poor driver.
- Insurance Companies
    - Insurance companies can use driver ratings and dashcam footage to adjust their rates and claims

1.6. Assumption And Limitations

Assumption:
The end product will be able to work under a certain degree of poor weather. It will not be able to work if the weather is too harsh, where the camera is not able to capture clear pictures. The end product will be used in the United States.

Limitations:
The end product will be a dashcam and a mini pc in the client's car, completing the Dashcam Defender apparatus. The cost to produce the end product shall not exceed one hundred and fifty dollars. The system will not require more than 12 Volts. (The most common voltage output found in an Automobile Auxiliary Power Outlet)

1.7. Expected End Product And Deliverables

Hardware Product

- The expected physical product is that we have a dash camera and a computational board that can communicate with each other
- The computation board is integrated with ALPR (Automated License Plate Recognition)
- The computation board can send data to a mobile device

Mobile Application

- An aesthetic mobile application
- A driver mode screen
- The application should be able to send data to a server
- User should be able to view information like cars encountered

2. Specifications and Analysis

2.1. Proposed Approach

Our proposed approach is to find existing resources that will help us fill in for things that we need. A mobile application will be developed utilizing the Flutter Framework. We chose Flutter because of its great layout design and versatility to create an android application and IOS application with one project. The camera and mini PC have to work together and the other half of the team will work on that. The mini PC we are going to use is the NVIDIA Jetson TX2. We chose the Jetson because a Raspberry Pi isn't powerful enough to run machine learning applications. The camera we are going to use is a standard 1080p camera. Our solution to analyze license plates is to use an open-source Automatic License Plate Recognition software called OpenALPR.

2.2. Design Analysis

Currently we have a mock database to test the connection between the hardware and the software. From here, we need to move into figuring out how to organize and send the information to fill the tables in the database. Our final version of the product will be doing this automatically, so we need to ensure the information can be processed efficiently. On the hardware side, our goal is to be able to read license plates at a rate of 5 per minute. On the software side, the program has to be able to multithread efficiently enough to process each license plate and store the information separately. The main strength of the project is in the database and the pushing and pulling of the data from it. The weakness, currently, is in the way we are pushing the data when the device doesn't have easy access to the internet.

2.3. Development Process

We are going to be following the Agile Methodology for our development processes. Specifically, we're going to split up into two separate teams of three; one team focused on software and the other focused on hardware, following the principles of Agile development. We think Scrum is the best choice for the development process because it will allow us to create small goals that are attainable as well as foster communication and collaboration within each group. We'll incorporate 2 week sprints because it aligns with the submissions of the bi-weekly reports. We will keep track of the work to be done in sprints in Trello.

2.4. Conceptual Sketch

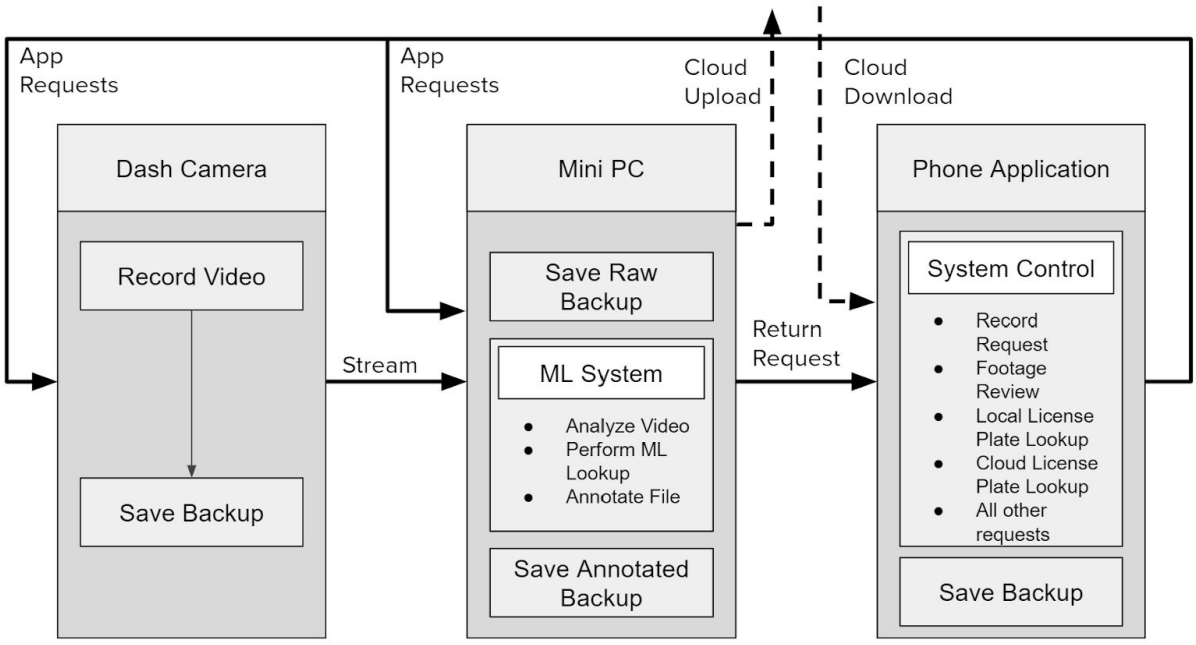


Figure 2.1: System Level Conceptual Sketch made by Evan Timmons

3. Statement of Work

3.1. Previous Work And Literature

OpenALPR - We're utilizing this service for license plate recognition because it has an acceptable pre-trained model on license plates.

3.2. Technology Considerations

Strengths

- Computer power on NVIDIA Jetson TX2 is adequate for our use case
    - 256-core NVIDIA Pascal™ GPU architecture with 256 NVIDIA CUDA cores
    - Quad-Core ARM® Cortex®-A57 MPCore
    - 8GB 128-bit LPDDR4 Memory
    - 32GB eMMC 5.1 storage
- *OpenALPR* is accurate enough (99%) for our use case
- *Flutter* allows for speedy development with one single codebase
    - Maintains native performance

Weaknesses

- *OpenALPR* is not 100% accurate, so edge cases can be tricky
    - *OpenALPR* is self-rated to be 99% accurate
- Cellular service in rural areas is not sufficient for constant data uploading. It will only be able to upload the data once it receives a sufficient internet connection. This will be dependent on the company of mobile data.
- The camera quality will not be as good in bad weather conditions
    - e.g. rain, snow, hail, etc.

Possible solution

- Create our own license plate recognition system that is more accurate
    - Requires training our own model
    - This will take a large amount of time with a lot of data required
- Only upload data while connected to a reliable source
    - This allows devices with data caps to upload only when connected to Internet

3.3. Task Decomposition

Hardware

- Hook external camera to *NVIDIA Jetson*
- Connect *NVIDIA Jetson* to *OpenALPR*
- Connect *NVIDIA Jetson* to mobile application
- Test runs

Software

- Develop *Flutter*-based mobile application
    - The mobile application is for the user to be able to connect their phone to the mini-PC to be able to interact with each other. The mobile application accesses the information about drivers as well as sends information to a server.
- Develop PostGRES database

    - Set up API to connect to database
    - Create security roles
    - Talk with subteams about what to store
    - Create high level diagram
    - Allow API to have public access
    - Create security when making api calls
    - Work with subteams to ensure POST and GET calls work properly
- Connect mobile application with *NVIDIA Jetson*
    - Connecting the mobile application and the Jetson is to send information from the jetson to the phone, then for the application to send or store information from the PC. It accesses the dash camera via the mobile application.

3.4. Possible Risks And Risk Management

Distracted Driving

- Use of mobile application while driving can be distracting to the driver
- To manage, we are utilizing a simple interface to minimize distracted driving

Hardware Compatibility

- *NVIDIA Jetson's* built in camera is not compatible with *OpenALPR*
- To manage, we're hooking up an external camera instead

Privacy Concerns

- The data we are working with is personal and will stay personal
- To manage, we are going to have a clear and robust data management policy
- There is a risk that user personal information will be stolen. We secure the personal information by having each user have their own account, which requires a unique username and password. This will be able to protect the privacy of users.
- We will prevent users from searching other user's footage. We will only allow users to search for the rating of other users, or the last location of the user if the user allows us to show their location.

Technical Risks

- We don't have any experience in machine learning. Developing machine learning that recognizes car plates might cost us some time and resources to learn on it. We will encounter this issue by spending more time to learn it online.

Software Risks

- If this mobile application has too many users using it at the same time, the database might be overloaded. We need to fix this by upgrading the server to be capable for all users to use it at the same time.
- Our team doesn't have much experience in developing software in Flutter or using the Dart language. We will learn it as we go.

3.5. Project Proposed Milestones and Evaluation Criteria

Key Milestones (in chronological order)

- Hardware Assembly completed
    - *NVIDIA Jetson* and external camera hooked up successfully
    - Designed and built enclosure for hardware
    - Completed power source for in-vehicle use
- Database Setup completed
    - Created initial test tables
    - Understood database design based on subteam requirements
    - Created users on the virtual machine to make database calls
    - Created API to speak with the database
    - Made API public so that that different subteams can access it
    - Began integration with the mobile app
    - Began stress testing the api through the app
- Mobile application completed
    - *Flutter* based mobile application with all requirements ready to go
    - A driver mode option for the safety of the driver
    - Is able to send information to the database/server
    - The application is able to receive information from the server successfully
    - IOS and Android mobile application are both working successfully
- *OpenALPR* working with *NVIDIA Jetson* w/ camera
    - Able to successfully feed video data into *OpenALPR* and detect license plates
    - OpenALPR tested to ensure maximum performance achieved based on hardware
    - Able to analyze 720p video at full 60fps
- Web Application
    - Fully functionally angular prototype
    - Search function to filter through 1000+ available videos
    - Search time of less than 2 seconds
    - Multi user capacity of 100 simultaneous instances
    - Testing suite implemented with Jasmine and Karma
- Complete working product
    - Able to interface with recorded data inside mobile application

3.6. Project Tracking Procedures

- *GitHub* for version control
    - 7ombers of our team will be doing work on separate branch
    - Code will be reviewed before merging
- *Trello* for backlog and tasks to be done during bi-weekly sprints
    - Trello will help with keeping track of progress of task
    - Task will be divided into section of To-do, in-progress and finished task
    - Each task will have a description and checklist work what need to be done to complete the task
    - Each task will be assigned to team member
- *Google Drive* for document sharing

- Document that need to be stored in a google drive folder that is accessible for every team member
- Everyone will be able to edit the document

3.7. Expected Results and Validation

- Fully functional hardware consisting of *NVIDIA Jetson* w/ camera that interacts with our *Flutter* based mobile application and *OpenALPR* to successfully read license plates from a user's car ride.
- Going to verify these expected results with real life examples.
- Mobile application will be verified by generating an android and ios emulator to run on it. (see section #6.3 (figure 6.1~ 6.5) for more details on the mobile application User interface) These UI are the most important UI. We might still need some other UI to make our mobile application better. For example, UI for users to rate other vehicles, to search for other vehicles, and settings.
- Web application result (see section #6.3 (figure6.6~ 6.10) for more details on the mobile application User interface)

4. Project Timeline, Estimated Resources, and Challenges

4.1. Project Timeline

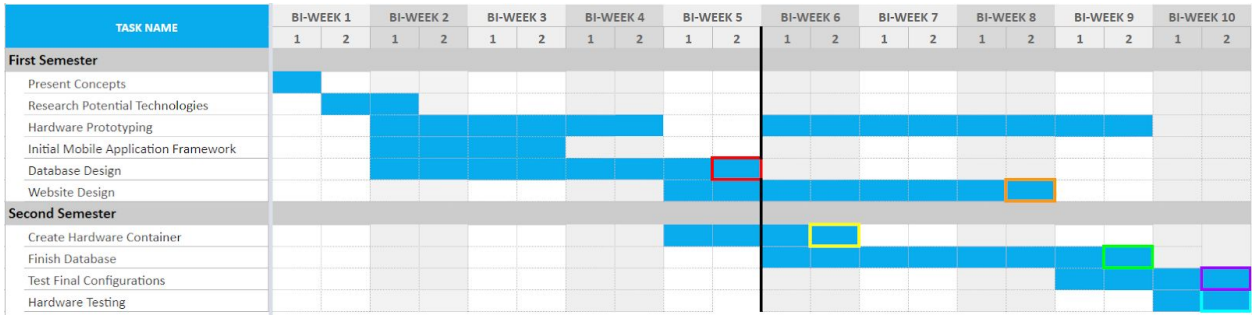


Figure 4.1: Timeline of milestones

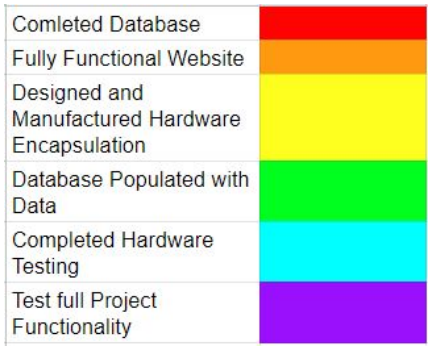


Figure 4.2: Color keys

Our Gantt chart for the project is shown above. In column one, the tasks are divided into two subsections, one for each semester. Tasks are mainly grouped by the semester of which they

begin. We highlighted the key deliverables with unique colored outlines and explained them right below the chart.

Uniquely, our project has been hindered by the COVID19 pandemic. As a result, our tasks have been divided to account for the inability to host physical meetings. Notably, hardware prototyping was put on hold until our team is able to meet again next semester as we are unable to order new parts to continue the prototyping phase.

4.2. Feasibility Assessment

For a completed project, we would like to have the mobile application, web application and hardware completely developed and tested. Due to a variety of factors, mainly time and access to hardware, this is not likely feasible. Our hope is that we are able to complete enough of each part of the project to have a working prototype with all core features. However, it is likely that not every component will be completed. The software part (mobile application and web application) are more feasible because we can distribute work, and work on the different parts, then merge it back. We just needed to have more virtual meetings, and we did not need any equipment from the lab. While the hardware is not feasible because we need equipment, and we would need to work on it in the lab. We can't work in the lab, and we can't meet face to face to work on the hardware part due to COVID-19.

4.4. Personnel Effort Requirements

Tasks to be Completed	Estimated Time to Complete
Present Concepts	The presentation of concepts only took our team 1 week to complete and was done during the first bi-weekly period
Research Potential Technologies	Researching technologies took our team a week to complete and another week to order the hardware components we had decided on.
Hardware Prototyping	Hardware prototyping is estimated to take 14 weeks to complete over two semesters. This task will involve creating a prototype from the NVIDIA Jetson TX2 and testing it in the environment of a vehicle.
Initial Mobile Application Framework	Initial mobile application framework took 4 weeks to complete and allowed our software team to get a better feel of Flutter.
Database Design	Database design began our second biweek and will continue for 8 weeks until biweek 5. The main goal will be to establish communication between both the mobile app and the web app.
Website Design	Website design is estimated to take 8 weeks over two semesters. The web app team will learn angular and create a web app that will allow users to view recordings from driving sessions and search license plate numbers to find specific recordings.
Create Hardware Container	Our hardware container will take us approximately 4 weeks to do and will account for stress testing of the finished product. We will need to research more into what materials to use for the enclosure so that there are no hardware performance issues as well as determining the proper dimensions the enclosure needs to be.
Test Final Configurations	Testing our final configurations is estimated to take around 8 weeks during only the final semester. Testing final configurations will consist of testing usability and load testing of the mobile application and web application. This will include making sure the user experience is solid and that there are no performance issues or bugs within either application.
Finish Database Design	Finishing the database will take us 8 weeks to complete, beginning semester 2, and will take into account memory management and speed of getting

	information.
Hardware Testing	Hardware testing is estimated to take 2 weeks only during the final semester. Testing will consist of ensuring that OpenALPR is working on the NVIDIA Jetson TX2 and that the camera is recording driving sessions at 1080p and sending those recordings to the database.

Table 4.3: Requirements for personal efforts

4.5. Other Resource Requirements

The costs needed for these requirements are further detailed in section 4.3

Resource Requirements	Comments and Alternatives
Fuel	While we have vehicles that we can use for testing free of charge, we will have to pay for the fuel costs that will be accrued after driving for miles of testing
Driver	This can be anyone on the senior design team who holds a valid driver's license
3D Printed Components	The parts should not be a large expense as we can pay for the price of the materials used.
OpenAlpr	This software is open source and has been lightly tested by the team. There is a higher performing paid alternative that could be used.
NVIDIA Jetson TX-2	OpenALPR site provided benchmarks with the Jetson which we think meets the specifications of our project. Based on benchmarks we should be able to run OpenALPR up to 30 fps on this board, and it also has all I/O needed (WiFi, bluetooth usb etc)
Sandisk 256GB SD Card	OpenALPR is typically not throttled by storage speed, this SD card provides a large amount of memory for testing, while also maintaining a cheap price
Unzano HD Webcam 1080p	A usb camera that will plug and play with Ubuntu and will work with OpenALPR software. The camera is 1080p (required for project) and has long distance manual focus (uncommon for webcams and required for project)
Logitech K400 Plus Wireless Touch TV Keyboard	Combo keyboard and mouse that can be used in a vehicle for prototype testing
Ultra-thin Monitor 7 Inch HD Portable Display Screen	Small monitor that can be used for prototyping and to emulate the product within the car.

4.6. Financial Requirements

There are many different hardware components and peripheral devices that are required for our project to be successful. Using the ETG to order these parts, we have come to the conclusion that the total financial resources required for our project is $450. This includes parts such as the NVIDIA Jetson TX2, a 1080p web camera, a SD storage card, and display monitor.

Resource Requirements	Cost
Fuel	~$50
Driver	$0
3D Printed Components	~$30
OpenAlpr	$0
NVIDIA Jetson TX-2	$299
Sandisk 256GB SD Card	$54
Unzano HD Webcam 1080p	$25.99
Logitech K400 Plus Wireless Touch TV Keyboard	$24.97
Ultra-thin Monitor 7 Inch HD Portable Display Screen	$45.99
Total Costs (estimate)	~$530

5. Testing and Implementation

5.1. Interface Specifications

We are going to use black box testing on our system if some part is not complete yet.

To test the mobile application we are going to develop unit tests using Flutter's own unit testing library. To help us out with mocking dependencies, we are going to use the Mockito library that is compatible with Flutter.

Our hardware development was cut unexpectedly short by the COVID-19 pandemic. However, we did manage to run OpenALPR, an open source software that will be used in our project, on the board. Eventually, the development board, an NVIDIA Jetson TX-2, will have to interface

with our backend and mobile application. That task has been deferred to next semester when we are able to continue with our hardware development.

5.2. Hardware and software

Software

We will decide to choose the best testing hardware and software once we have some part done. Flutter test library is a package used in flutter to make test cases; It can be used to make unit tests. It can be used for many functionalities on a mobile application. Mockito is a testing framework. The framework allows us to mock dependencies rather than fetching them from a database.

Database

We also use Postman to test the database by getting and posting data to it. This will allow us to make sure our database is working well before the front end is accessing it.

Web Application and Hardware

For testing or angular web application, we will be using the Jasmine testing framework. This will allow us to continuously test functionality and appearance as we add more to the web application. For hardware testing the NVIDIA Jetson TX-2 we have nothing currently planned for testing but will research more into it once we have access to the device again.

5.3. Functional Testing

We are going to test all the functionalities for our mobile and web application.

We will test our mobile application with unit testing in flutter. We will do some automation testing.

We are building our web application using the Angular framework. The framework divides web application components into “modules” with interconnected functionality. We have functional unit tests for each module that has been implemented. Through the Angular CLI we are able to frequently run our unit tests using the built in Karma test runner.

We are using SQL and python to create our backend. To test this, we have installed a framework called python virtual environment. This lets us deploy the database to a local port. From here, we can run various commands through an API testing app, such as Postman, and ensure the database is read from and populated correctly. We followed the iterative design approach outlined in [6].

5.4. Non-Functional Testing

Security - We are having a login page, which requires a username and password for the user to login. The passwords will be encrypted. It will also be able to connect with Facebook accounts.
Performance - We’ve launched our mobile application in Android and iOS environments. Currently, they run smoothly and it is doing what we are expecting.

Usability - Users are able to use it like a normal mobile application, which prioritizes user experience while not skipping out on functionality.
Compatibility- We designed the mobile application to work on Android and iOS. We have tested it with a few android and iOS emulators.

For the web application there are many non-functional tests that we are considering as well. First off is the performance of the application. To do this we will attempt to create load tests to create high demand on the application and test its response times. Another test we must consider is usability. To test this we will have usability test sessions where we can express our concerns with the user experience To test for compatibility we will make sure that the web application looks and navigates the same on multiple web browsers/environments. Finally, to make sure our web application is secure we will require two factor authentication.

5.5. Process

Mobile App

We will be testing the Mobile Application with Flutter's built in unit testing library, named the Flutter Test Library. This allows us to test the functional requirements of the app. To test aspects that require us to use data from our database, we will use Mockito to mock dependencies over fetching real data from our database, making testing quicker.

Flutter has a built in Hot Reload feature that allows for real time compilation of code into functionality at the press of a button. This speeds up our testing workflow greatly and allows the team to develop features quicker and debug quicker, especially for functional testing.

Web App

We will be testing the Web Application with the Jasmine testing framework. Jasmine has the Karma testing interactive interface built in for more seamless testing procedures. We have yet to begin testing the web application, as it is not yet complete. When we do begin testing, we will focus on the search result speed, aiming for below 2 seconds of time. Additionally, we want to test the Web App with upwards of 1000 videos to ensure that the accompanying features still work properly. Finally, we want to implement multi user testing, confirming that 100 users can load the application and watch a clip at the same time.

Database

For the database, we have begun initial testing of the API by opening it publically such that the mobile app can access it. This will ensure the API can go out to the database and make requests accordingly. From here, we moved to initial stress testing of the database by making several calls to the database every hour. From here, we will redo this process once the Webapp is ready to receive information through the api. We plan on expanding this so that the api can send and receive video information.

Hardware

As mentioned in an earlier section, we will be testing hardware by ensuring that OpenALPR is installed and configured correctly on the NVIDIA Jetson TX2 and that it is

recording driving sessions in 1080p. We will also test to make sure that the database is configured to receive these videos from the hardware.

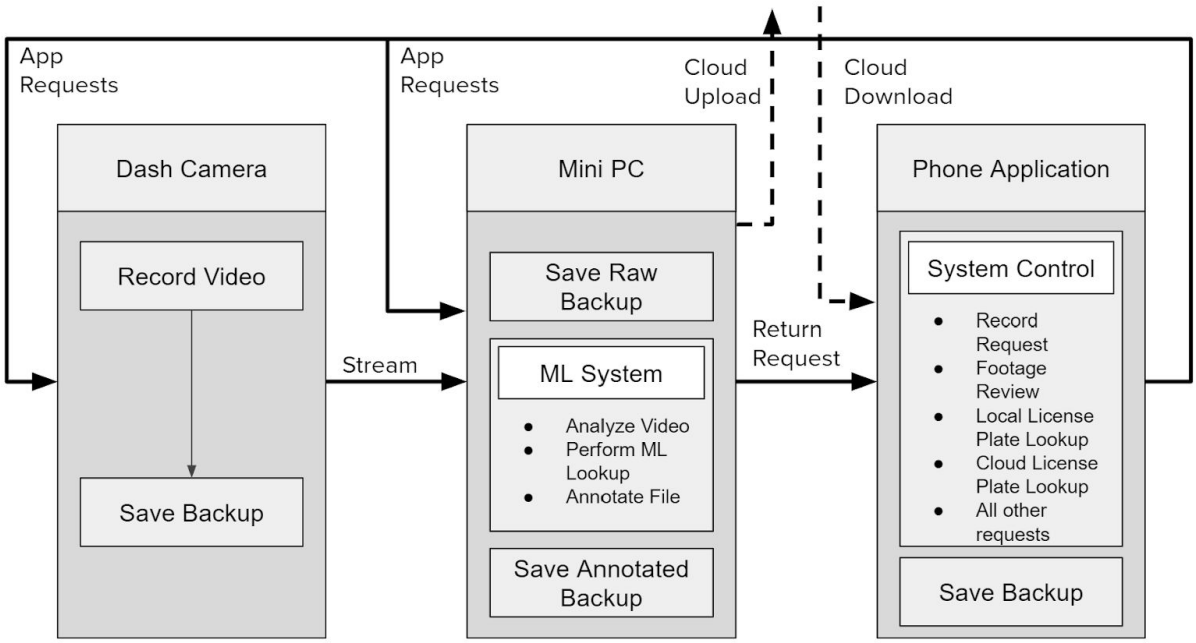


Figure 5.1: Process Flowchart

5.6. Results

Given the structure of our development, our results will be obtained during our 492 semester. At this time, we are only able to discuss how we will collect and display the results that we do end up achieving, and speculate upon the efficacy of our development.

Failures

- Facebook login page in Flutter is failing to run on iOS.

Successes

- Facebook login page is working when we run it on an Android device.

What we learned, and how we plan to change it

- For the software team, we have learned how to work on flutter by using .dart language. We might add a sign in with google button, or we might remove facebook login button if we are not able to resolve the issues that it is failing to run on IOS.

Modeling and Simulation

- We will add modeling and simulation information next semester once it has been completed.

Issues and challenges

- Our team has no experience in flutter and dart.
- We are not being to meet by person to discuss about the project due to the virus

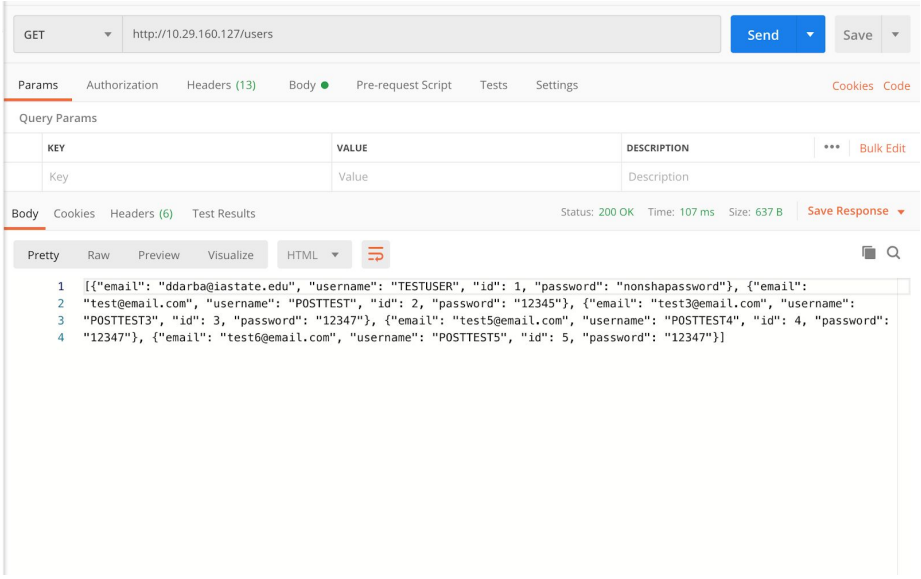
The hardware team is in a similar position to the software team in that the majority of our results will be obtained during the 492 semester.

Failures

- Inability to configure the built-in camera on the NVIDIA Jetson TX-2

Successes

- Over the course of the semester, we have been able to set up the virtual machine with a Postgres database to store information.
- We've created a basic api that allows different subteams to begin integration to the project as a whole.
- Screenshots of test data being sent and received from the database



e:

Figure 5.2: Screen capture from Postman pulling user data.

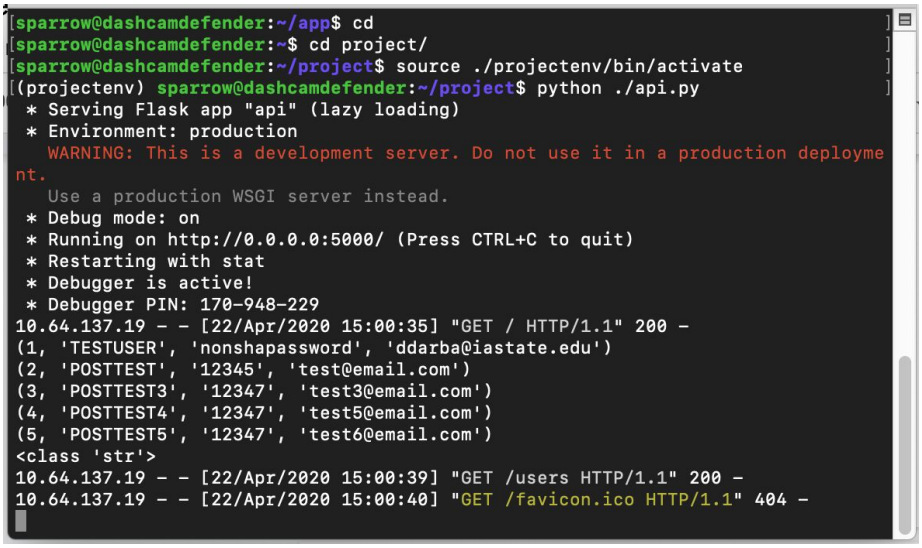


Figure 5.3: Screen capture of the virtual machine reporting each API call.

Information Learned

- We are beginning to learn Angular to develop our web application and have had success setting up the database.

Issues and Challenges

- Inability to meet in person to test and configure the NVIDIA Jetson TX2
- Inexperience with developing in Angular

6. Closing Material

6.1. Conclusion

In order to help drivers on the road, we need a better way to hold reckless drivers accountable. With our app and dash cam, we can do this. Furthermore, we can help insurance companies and the police get a better idea of people on the road. By meeting our specifications through the milestones we've set, we will be able to finish this project in a timely fashion.

6.3. References

General Internet Site

[1]

Codesundar, “Flutter Facebook login with Example,” *codesundar*. [Online]. Available: https://codesundar.com/flutter-facebook-login. [Accessed: 18-Apr-2020].

Developer Documentation

[2]

“Hot reload,” *Flutter*. [Online]. Available: https://flutter.dev/docs/development/tools/hot-reload. [Accessed: 18-Apr-2020].

Official Agile Manifesto

[3]

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jefferies, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, “Manifesto for Agile Software Development,” *Manifesto for Agile Software Development*, 2001. [Online]. Available: https://agilemanifesto.org/. [Accessed: 18-Apr-2020].

Journal Article in Scholarly Journal (published free of charge on the Internet)

[4]

S. Du, M. Ibrahim, M. Shehata, and W. Badawy, “Automatic License Plate Recognition (ALPR): A State-of-the-Art Review,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 2, pp. 311–325, Jun. 2012.

General Internet Site

[5]

"Setting up the local environment and workspace," *Angular*. [Online]. Available: https://angular.io/guide/setup-local. [Accessed: 15-Apr-2020].

Journal Article in Scholarly Journal (published free of charge on the Internet)

[6]

S. W. Ambler, "Test-Driven Development of Relational Databases, " IEEE Software, vol. 24, no. 3, pp. 37 - 43, May-June 2007.

6.5. Appendices

Any additional information that would be helpful to the evaluation of your design document. If you have any large graphs, tables, or similar that does not directly pertain to the problem but helps support it, include that here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc. PCB testing issues etc. Software bugs etc.

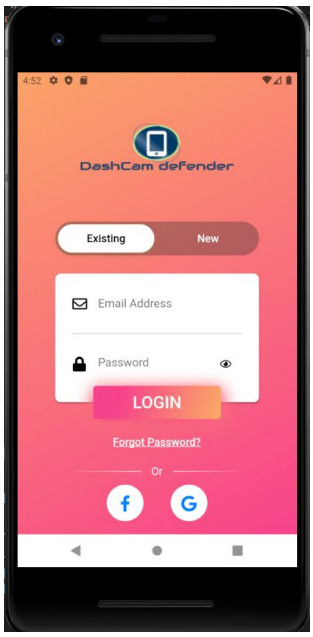


Figure 6.1: Login page

Figure 6.2: Car selection page

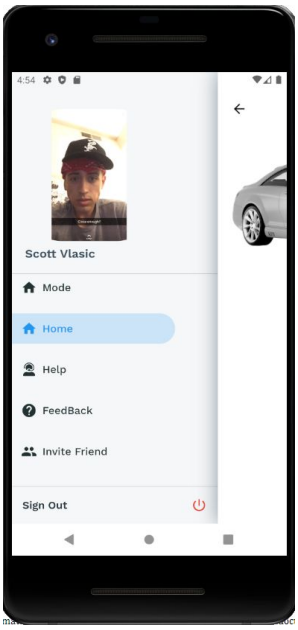


Figure 6.3: Sidebar

Figure 6.4: Mode Selection Page

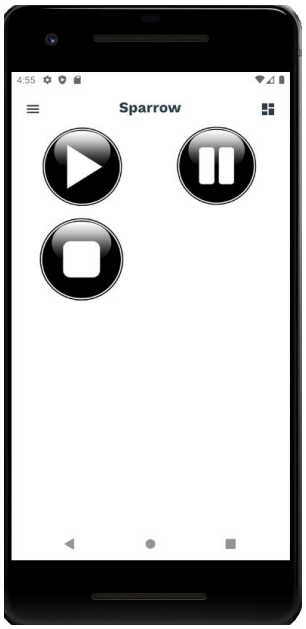


Figure 6.5: Driver Mode page

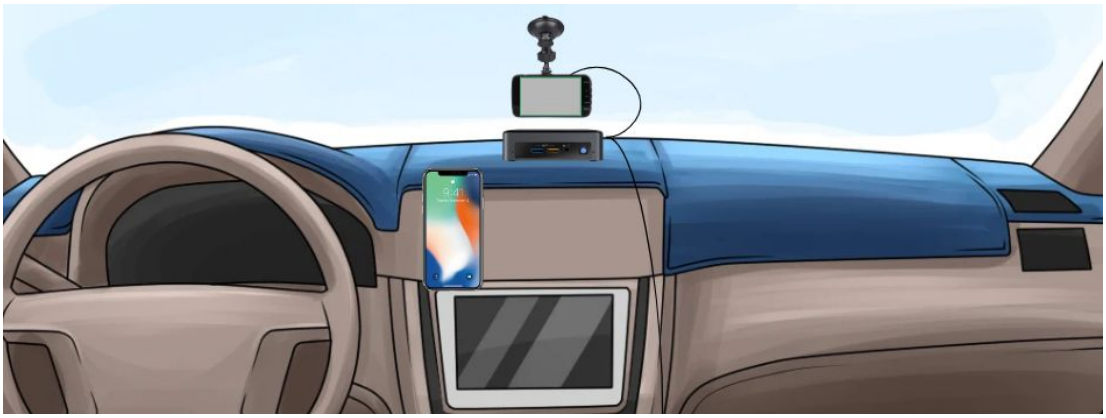


Figure 6.6: Concept image of full system in a vehicle

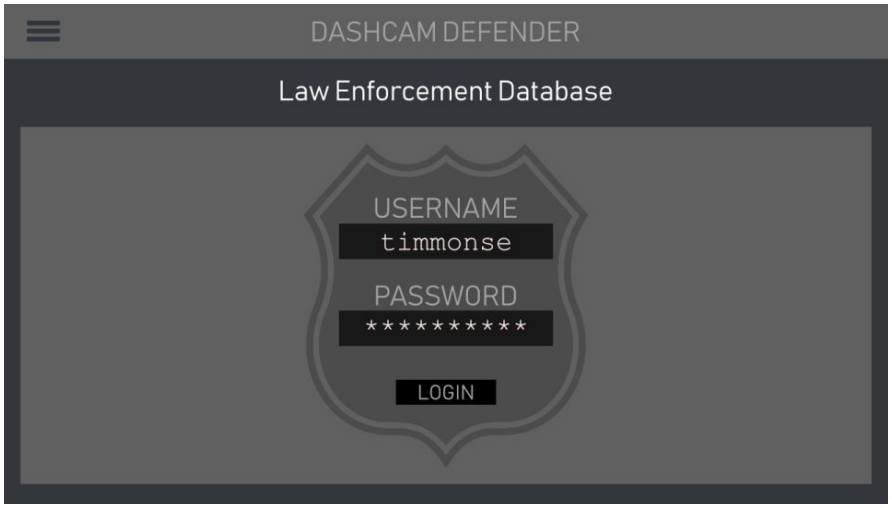


Figure 6.7: Prototype law enforcement database login

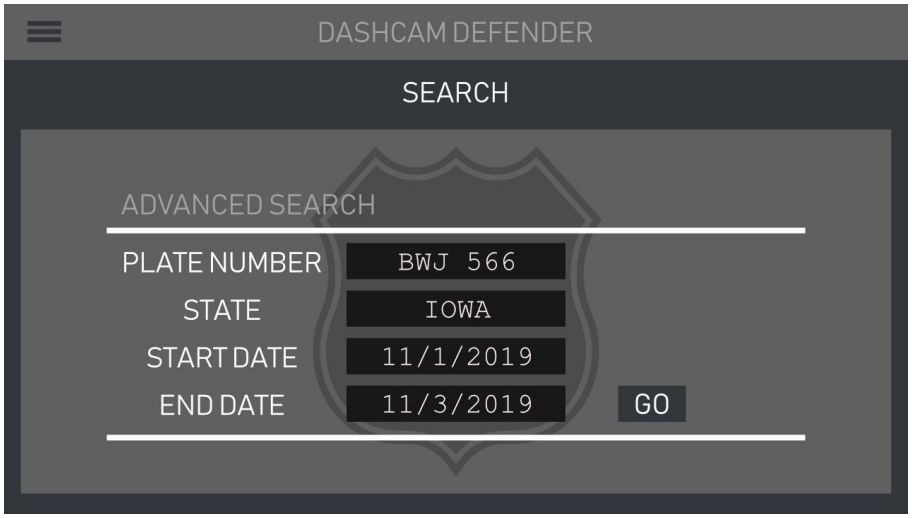


Figure 6.8: Prototype law enforcement license plate search

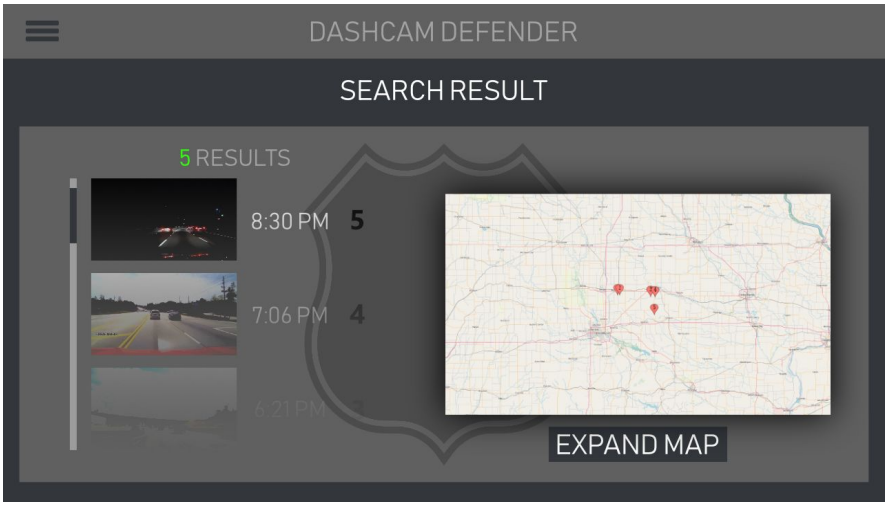


Figure 6.9: Prototype law plate search results

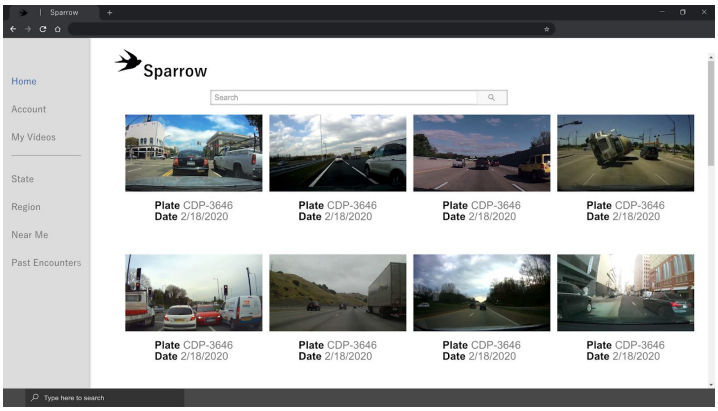


Figure 6.10: Prototype WebApp

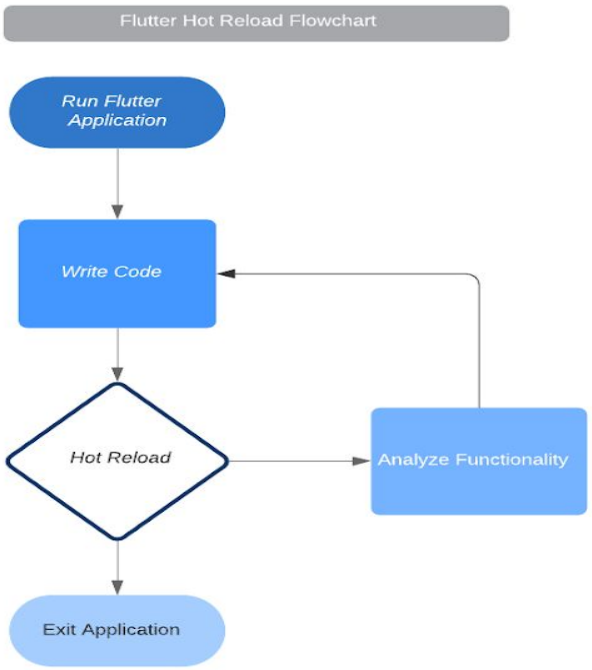


Figure 6.11: Hot Reload Flowchart

Software bugs- Mobile applications are working as normal in android phones, but having issues while launching the mobile application for IOS.

List of Figures

Only figures used so far, made by Evan Timmons. Used in section 2.4 “Conceptual Sketch” Additional figures will be cited here when used

10. Appendix III: Other Considerations

- **ALPR System Enclosure**

  In the next phase of our development, the team would continue working on a much safer enclosure for the NVIDIA JETSON board. With our current setup, there is a lot of room for the board to get spilled on or damaged in a car accident, so our next course of action would be to 3D print a casing for it.

  We would do this by first figuring out which part of the NVIDIA JETSON are absolutely necessary and which parts aren't. From here, we would custom build the PCB and take out any unnecessary space. This would ensure that the user experience would be clean cut and the board itself wouldn't be a safety hazard. Finally, we would design a casing around the board that protects it from the elements while allowing complete visibility of the road.

- **Law Enforcement API System**

  If development were to continue, the Sparrow team would create an API system to be used by law enforcement agencies. A system like Sparrow could be incredibly useful to find and prosecute criminals and potentially save lives. As alluded to earlier in the document, the system could be used to identify Amber Alert vehicles in real time and notify authorities of their location. Additionally, the system could be used to find stolen vehicles, get away vehicles, and potential convicted felons.

  As discussed in the context portion of this report, the power of this software comes with valid privacy concerns. The Sparrow team takes these concerns very seriously and will implement measures to protect users' privacy. For instance, this Law Enforcement API would only grant access to certain functionalities and most requests would have to be approved. To maintain full transparency, all requests for data would be sent to the applicable registered vehicle owners for approval. That way, in the hopefully infrequent cases where law enforcement uses the system against better judgements, the vehicle owners are aware and can deny data requests.

  In the case where the registered vehicle owners are the criminals being investigated, Sparrow would require law enforcement to obtain a warrant before obtaining access to the vehicle data.

- **Insurance API System**

  If project development was continued, then the team would look to implement an insurance system API to be utilized by insurance companies to assist with claims. This system would be useful for determining the cause of blame in an accident and the footage could dispute or support claims made in court. Insurance companies would also be less susceptible to fraudulent claims as they would have the video footage available.

  As with the law enforcement API, the insurance system API draws concerns of privacy among users. The Sparrow team would ensure that insurance companies are not able to discriminate against drivers based on the footage that they receive and that user rates are dependent on the policies of the company.

- **Flutter mobile application**

  In the future, the software team would continue to implement more functionalities, and more utilities. Our main goal is to make sure we have enough resources to maintain the mobile application, and continue integration with the ALPR system, so that the mobile application can

work more efficiently. In addition, we can have functionality to control the user's camera to start recording and stop recording, controlled by the phone remotely.

Furthermore, we can add the functionality that users can watch other people's videos, if only if they have the privilege. We would also need to implement a stronger security feature into this mobile application. This is because the video information is confidential, and it can be used in many ways. We might need to have Multi-factor authentication, and have a strong encryption, like AES 512 encryption to store user information in the database. This will definitely make user's information more secure.

11. Appendix IV: Code

- **Android/iOS Application**
  - https://github.com/cobijm/Sparrow
- **AWS Lambda Functions**
  - https://github.com/cobijm/SparrowLambdas
- **Embedded GUI**
  - https://github.com/timmonse/sparrow-local-gui

12. Additional Documentation

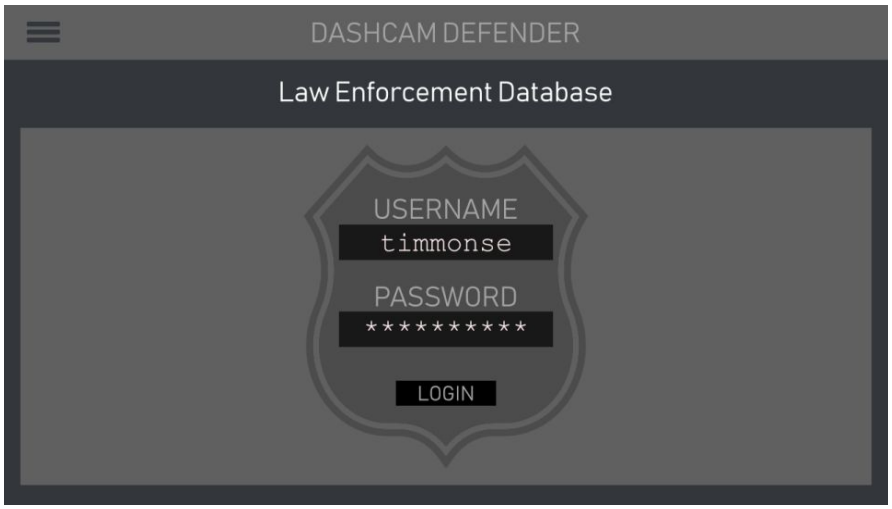


Figure 12.1: Prototype law enforcement database login

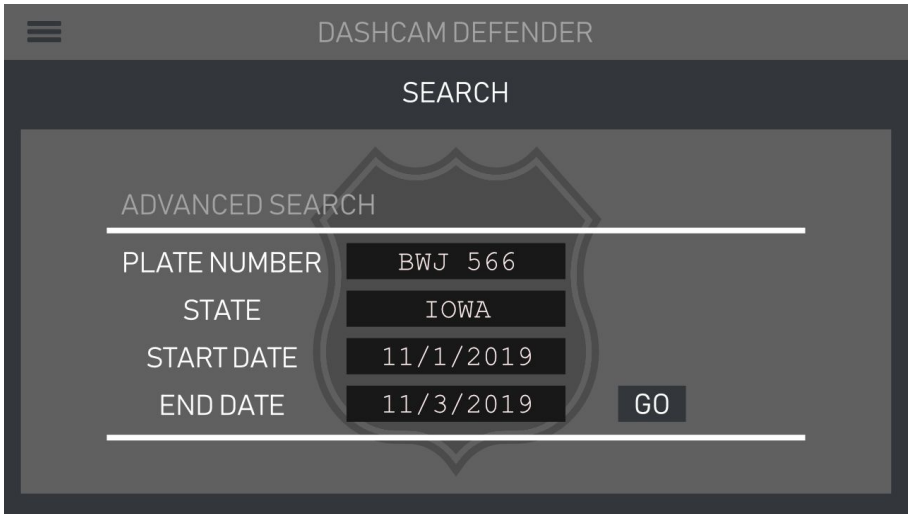


Figure 12.2: Prototype law enforcement license plate search

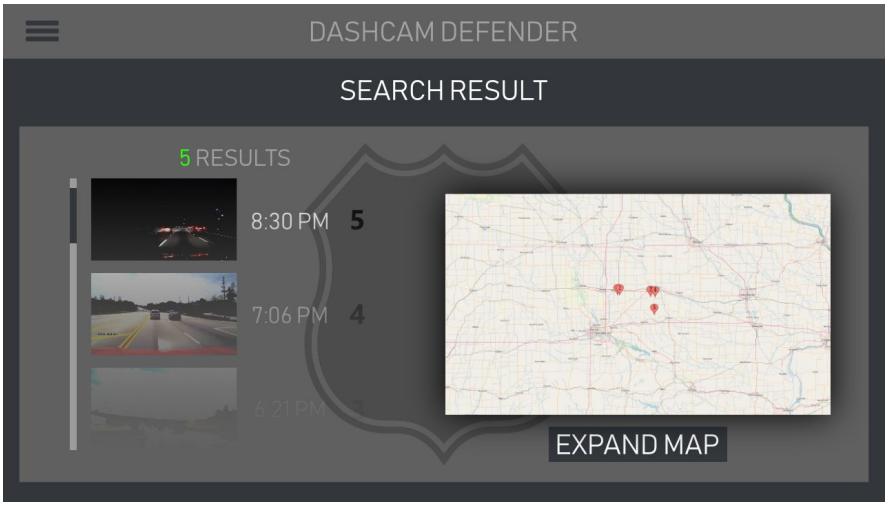


Figure 12.3: Prototype law plate search results

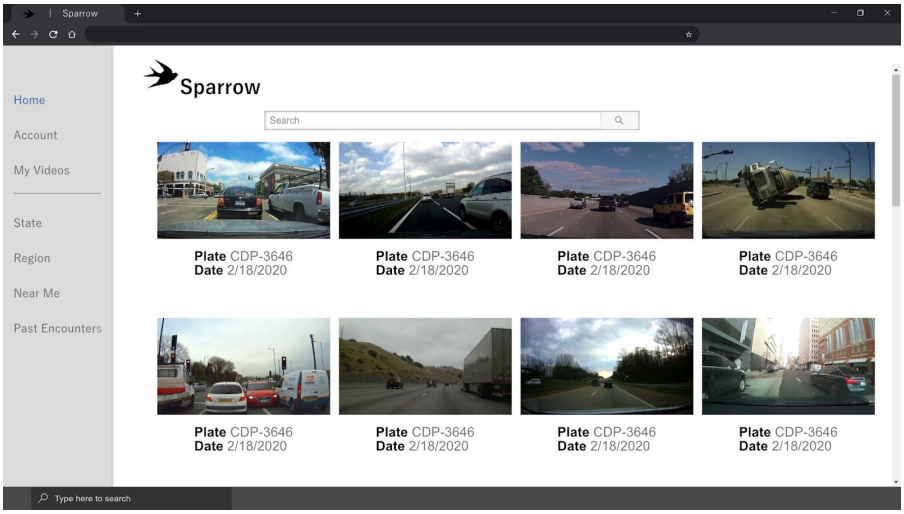


Figure 12.4: Prototype WebApp

Figure 12.5: Prototype image 1

Figure 12.6: Prototype image 2

Figure 12.7: Prototype image 3

Figure 12.8: Prototype image 4

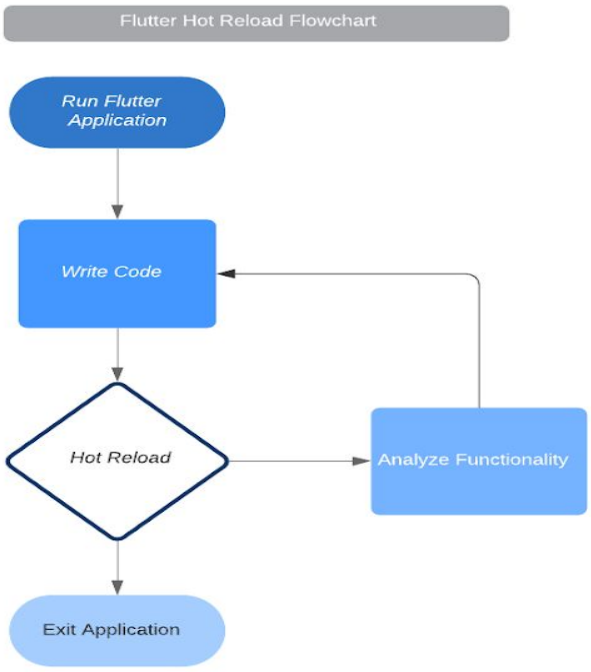


Figure 12.9: Hot Reload Flowchart

Software bugs- Mobile applications are working as normal in android phones, but having issues while launching the mobile application for IOS.

13. List of Figures