**Final Project Report for INDOT Traffic Monitoring System**

By

Holden Wright

Kyle Cooper

Kaleb McArdle

Ian Kenney

Samuel Akinmoyeje



Transportation Active Safety Institute • Dr. Stanley Chien • [schien@iupui.edu](mailto:schien@iupui.edu) • (317) 274-2760

ECE 48800 FALL 2019

March 22nd, 2019

**TABLE OF CONTENTS**

1. Project Motivation………………………....………………………………………………6
2. Background Information…………………………………………………………..............6
3. System Block Diagram……………………………………………………………………7
4. Project Requirements……………………………………………………………...............7

4.1 Technical Requirements…...............................................................................................7 4.2 Standards Requirements….............................................................................................13

4.3 Safety Requirements…..................................................................................................13

4.4 Constraints….................................................................................................................13

5.  Design Options………………………………………………………………………………..14

5.1 Option 1……………………………………………...................................................14

5.2 Option 2……………………………………………………………………………...16

5.3 Option 3……………………………………………………………………………...18

5.4 Subsystem 1………………………………………………………………………….19

5.5 Subsystem 2………………………………………………………………….............20

5.6 Subsystem 3………………………………………………………………….............21

5.7 Software information diagram……………………………………………………….23

5.8 Database block diagram……………………………………………………………...24

5.9 Implementation details……………………………………………………………….25

5.10 Implementation results……………………………………………………...............27

5.11 Safety……………………………………………………………………………….13

  5.12 Standards……………………………………………………………………………13

5.13 Constraints (Satisfied)……………………………………………………................13

7.  Project Plan…...........................................................................................................................30

8.  Testing Procedures…................................................................................................................30

9.  Conclusion and future work ………………………………………………………….............32

10.  List of References…………………………………………………………...........................33

11.  Appendices ………………………………………………………….....................................34

**TABLE OF FIGURES**

Figure 1……………………………………………………………………………………………7

Figure 2…………………………………………………………………………………………..15

Figure 3…………………………………………………………………………………………..16

Figure 4…………………………………………………………………………………………..18

Figure 5…………………………………………………………………………………………..19

Figure 6…………………………………………………………………………………………..20

Figure 7…………………………………………………………………………………………..21

Figure 8…………………………………………………………………………………………..23

Figure 9…………………………………………………………………………………………..24

Figure 10…………………………………………………………………………………………25

Figure 11…………………………………………………………………………………………28

Figure 12…………………………………………………………………………………………28

Figure 13…………………………………………………………………………………………29

Figure 14…………………………………………………………………………………………30

**LIST OF ABBREVIATIONS**

INDOT - Indiana Department of Transportation

TASI - Transportation Active Safety Institute

PHP - Personal Home Page

SQL - Structured Query Language

API - Application Program Interface

GUI - Graphical User Interface

UI - User Interface

IEEE - Institute of Electrical and Electronics Engineers

PTZ - Pan / Tilt / Zoom

**Project Motivation**

The Indiana Department of Transportation (INDOT) currently monitors traffic via operators who pan through several cameras searching for traffic congestion. As traffic increases over time it is becoming increasingly difficult to spot traffic abnormalities using the existing method. INDOT is asking the Transportation Active Safety Institute (TASI) for a faster way to spot and handle traffic issues. This system should reduce the load of work per operator by making various functionalities more convenient. The given task is to build a database and web-based Graphical User Interface (GUI) to assist the INDOT operators in finding and solving traffic issues at a faster rate than ever before.

**Background Information**

INDOT has asked TASI to help create an automatic vehicle detection system that will work in conjunction with INDOT’s existing camera infrastructure. INDOT currently has over 350 cameras in use, which must all be continuously monitored by human operators. This automatic vehicle detection should keep track of vehicle collisions, current weather conditions, number of lanes each road, traffic flow rates for each road/lane, and general traffic conditions such as traffic jams. To accomplish these objectives TASI desires a database to store information collected from the image processing side for every camera and a web-based Graphical User Interface (GUI) which will be used to view live feeds and statistics for every camera.

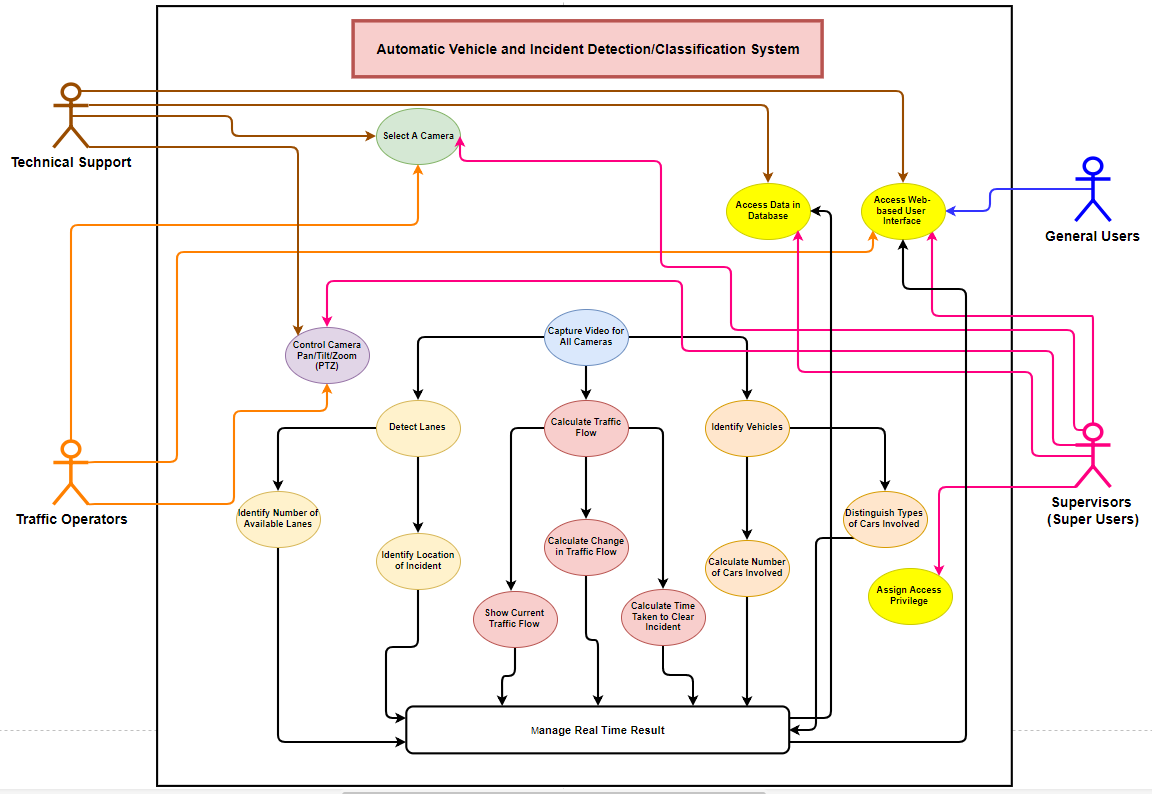
System Block Diagram  
  


Figure 1

**Project Requirements**

Original Technical Requirements

1. Database
   1. Shall read and present camera device information including:
      1. Camera ID, make, model, technical specifications
      2. Installation location information
      3. Maintenance/calibration information
      4. Default and preset angles
   2. Shall store camera associated road information on an information panel including:
      1. Camera ID
      2. Location ID
      3. Number of roads in each viewing angle
      4. Road type (road, ramp, etc., flyover)
      5. Speed limit of each road
      6. Number of lanes on each road
      7. Number of usable lanes
      8. The condition of each lane
   3. Shall store incidents to database according to:
      1. Date
      2. Time
      3. Camera ID
      4. Location ID
      5. Duration
   4. Shall store the traffic incident data into database including:
      1. Location of incident
      2. Lane numbers where incident has happened
      3. Direction of traffic
      4. Status – resolved or awaiting cleanup support
      5. The identity of involved vehicles
         1. Passenger Auto
         2. Mini Van
         3. Van
         4. Motorcycle
         5. Large Truck
         6. Truck
         7. Light Truck
         8. Bus
      6. The number of cars involved in the incident
      7. Current traffic flow for all lanes in the incident location
      8. Percentage of different car types
      9. Image resolution
      10. Start time of the incident
      11. End time of the incident
      12. Incident status
   5. Shall include Emergency response information including:
      1. Emergency response types
      2. Incident types
      3. Emergence response associated to incident types
2. GUI
   1. Shall provide login page
   2. Shall provide an administration page that can perform the following functions
      1. Add users
      2. Remove users
      3. Manage permissions
      4. Manage data
   3. Shall provide an interactive map including:
      1. Camera feeds
      2. Camera locations
      3. Camera Name
      4. Incident icons with priority coloring
   4. Shall provide sortable Incident information including:
      1. Location in GPS coordinates
      2. Incident categorized by the types and priorities of:
         1. Car Collision(Fatal): Highest Priority
         2. Obstacles on Road:
            1. Large Foreign Objects: High Response Priority
            2. Debris: Medium Response Priority
            3. Lane Closures for Utility Work: Low Response Priority
         3. Traffic Jam(Non-Critical):
            1. Peak Hours: Low Response Priority
            2. Presence of many Large Trucks or Semis: Low Response
            3. Extra Cars entering from ramp: Low Response Priority
            4. Car Collision: High Response Priority
      3. Time of incident
         1. Start Time
         2. End Time
         3. Total Duration
      4. Camera name
   5. Shall include a camera feed showing the active camera
   6. Shall include a camera control interface with functionalities of:
      1. Pan / Tilt / Zoom Controls (PTZ)
      2. Preset Directions for snap-to control
      3. Accept input from operator joystick

Based on the verification cross reference of the original requirements, many of them have not been totally satisfied. Although all of them have been addressed and are at least partially satisfied, there was no way to completely satisfy them without real incidents, real video feed, and complete image processing information. However, representatives from INDOT seem satisfied and pleased with the extent to which the original and updated requirements have been met. To account for the discrepancy between the original requirements and finished product, they have been amended below. Instead of shall statements, many of the requirements involving incidents have been changed to should statements. Furthermore, three features of the project that INDOT had originally requested have been redacted. This is explained in more detail in the Implementation Details section below.

Updated Technical Requirements

1. Database
   1. Shall read and present camera device information including:
      1. Camera ID, make, model, technical specifications
      2. Installation location information
      3. Maintenance/calibration information
   2. Should store camera associated road information on an information panel including:
      1. Camera ID
      2. Location ID
      3. Number of roads in each viewing angle
      4. Road type (road, ramp, etc., flyover)
      5. Speed limit of each road
      6. Number of lanes on each road
      7. Number of usable lanes
      8. The condition of each lane
   3. Should store incidents to database according to:
      1. Date
      2. Time
      3. Camera ID
      4. Location ID
      5. Duration
   4. Should store the traffic incident data into database including:
      1. Location of incident
      2. Lane numbers where incident has happened
      3. Direction of traffic
      4. Status – resolved or awaiting cleanup support
      5. The identity of involved vehicles
         1. Passenger Auto
         2. Mini Van
         3. Van
         4. Motorcycle
         5. Large Truck
         6. Truck
         7. Light Truck
         8. Bus
      6. The number of cars involved in the incident
      7. Current traffic flow for all lanes in the incident location
      8. Percentage of different car types
      9. Image resolution
      10. Start time of the incident
      11. End time of the incident
      12. Incident status
   5. Should include Emergency response information including:
      1. Emergency response types
      2. Incident types
      3. Emergence response associated to incident types
   6. Should include a construction flag to differentiate traffic and incidents from ongoing construction.
2. GUI
3. Shall provide an interactive map including:
   1. Camera feeds
   2. Camera locations
   3. Camera Name
   4. Incident icons with priority coloring
4. Should provide sortable Incident information including:
   1. Location in GPS coordinates
   2. Incident categorized by the types and priorities of:
      * 1. Car Collision(Fatal): Highest Priority
        2. Obstacles on Road:
           1. Large Foreign Objects: High Response Priority
           2. Debris: Medium Response Priority
           3. Lane Closures for Utility Work: Low Response Priority
        3. Traffic Jam(Non-Critical):
           1. Peak Hours: Low Response Priority
           2. Presence of many Large Trucks or Semis: Low Response
           3. Extra Cars entering from ramp: Low Response Priority
           4. Car Collision: High Response Priority
   3. Time of incident
      * 1. Start Time
        2. End Time
        3. Total Duration
   4. Camera name
5. Shall include a camera feed showing the active camera

Standard Requirements

1. Ethernet IEEE 802.3 Standards
2. Leaflet Map API standards
3. Internet Security protocol
4. RTSP: Real time streaming protocol

Safety Requirements (Security)

1. Compliance with existing active directory system
2. Care taken when assembling the server
3. Millions of highway traffic cameras are installed in roads and businesses throughout the world with the goal of reducing crime, while increasing accountability and public safety. This is a concern to those who value privacy. However, such a system can improve transportation reliability in general which could improve highway safety.

Original Constraints

1. Linux-Based Operating System
2. Database must be implemented with PostgreSQL
3. Webpage must be hosted on an Apache server
4. Working prototype within 1 semester (Satisfied)

Updated Constraints

1. Database must be implemented on IU’s network using MySQL workbench
2. Apache server must be hosted by IU’s network and use the program, webserve
3. Linux-Based Operating System

This constraint is satisfied. Development is taking place on a linux (Redhat) server.

1. Database must be implemented with PostgreSQL

Installed and began researching PostGreSQL. This constraint is on track to be satisfied.

1. Webpage must be hosted on an Apache server

Installed Apache to the server. When the website is complete it will be hosted on this server.

1. Working prototype within 1 semester

The schedule keeps the team on track to complete a working prototype by the end of this semester. Setbacks are being managed to keep the schedule current.

1. INDOT Active Directory

Schedule Requirements

Working prototype within 1 semester

**Design Options**

System Design Options

Option 1.

The design process derived from the project requirements given by INDOT. The starting point was creating a global reference point that filled every basic requirement. Figure 2 below is the original layout mockup which has been the foundation for later iterations.

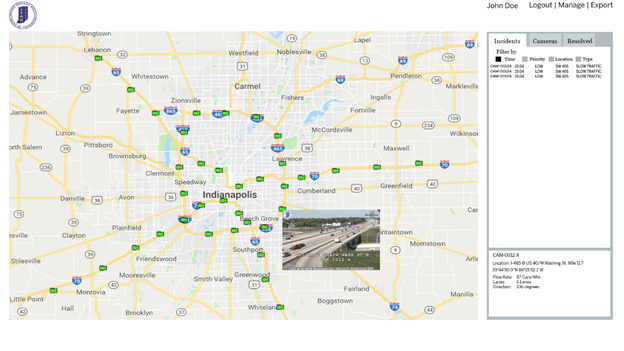


Figure 2

1. The original idea was to use the main reference point as the google map API with a hover toolbar. The above worked as so:
   1. On the map all the cameras are always visible and a default green color was chosen.
   2. Hovering over the camera causes the live feed will pop up and display.
   3. Upon right clicking the camera a list of options would appear as a toolbar.
   4. The right side was the incident list that would autofill anytime the database received a traffic issue.
   5. Sort by incident, camera, or resolved with filtering of each type of list by time, priority, location, and type.
   6. Contains user settings as well as admin settings in the top right corner.
      1. Logout – this would simply log the user out.
      2. Manage – this was the admin section where those with admin privileges could add/remove user, and do other requested admin features.
      3. Export – this would allow the user to export data for research and things of that nature.

Pros:

* Very simple User Interface.
* Covered the requirements.
* Easy access to Cameras.
* Easy way to handle issues.

Cons:

* Very basic, not pretty.
* Congested too much functionality into the map.
* Very hard to handle 300 cameras.
* Camera functionality is very limited using toolbars.
* The UI was very messy.

Decision: It was a very good start and was very useful as a reference point. The design needed a lot of polishing.

Option 2.

The next design was to take key functionality from the map API and make it its own part. Time was spent making the overall layout cleaner and more precise. The below mock-up is what was derived:

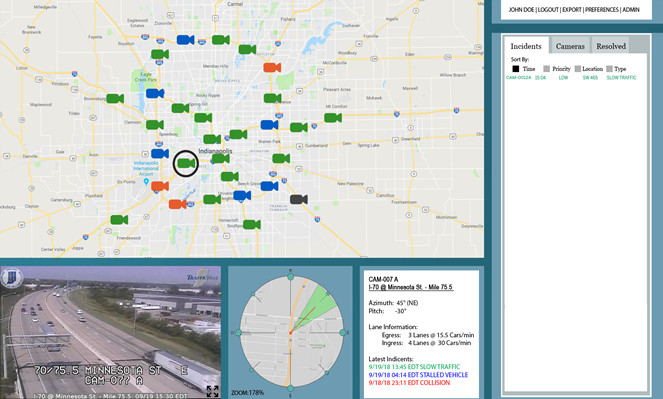


Figure 3

1. The above took the functionality from the original mock-up and did the following:
   1. Instead of showing the camera feed when hovering over the camera, it was moved to the camera feed to always display the previous selected camera on the bottom right.
   2. The addition of a camera angle location so that the operator would know the current area the camera was pointing at as well as the zoom level (depicted by the red line).
   3. The incident/camera info box was moved from directly underneath the incident list to right beside it. This would allow the operator to see basic info about the camera with the last few incidents displayed at the bottom.
   4. The map API cameras were changed to show a different functionality:
      1. If the camera had a low priority incident it would display green.
      2. If the camera had a medium priority incident it would display blue.
      3. If the camera had a high priority incident it would display red.
      4. If the camera had an issue it would display as black.
   5. The User/Admin settings were changed to make them more polished.

Pros:

* Maintained a relatively simple design.
* Improved functionality for cameras and operator access.
* Allowed operators to identify traffic issues in multiple different areas.
* Gave a more User friendly way to identify the issues.
* Allowed the operator to read and edit traffic issue information as well as make notes.

Cons:

* Still very bland looking.
* Still contains holes within the design.

Decision: The final design is getting closer and this will become a new reference point for an improved design basing most of the next mock-up on this design with only few changes.

Option 3.

The next and most current mock-up took the better of the previous two designs and created a better layout as shown below.

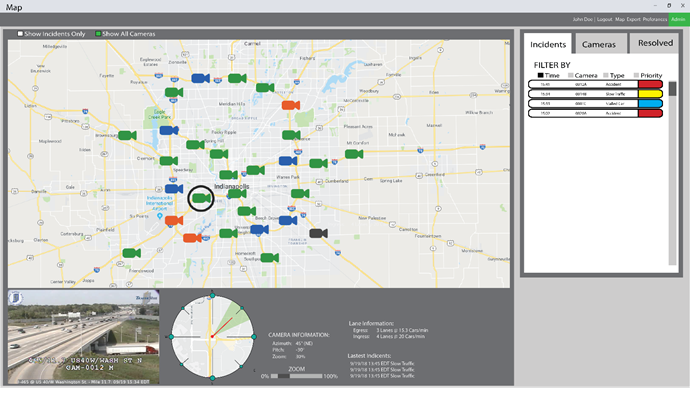


Figure 4

The above has added more functionality and made the overall program more user friendly with little to no pros and cons thus far.

Design Details

Subsystem 1

Compass controls

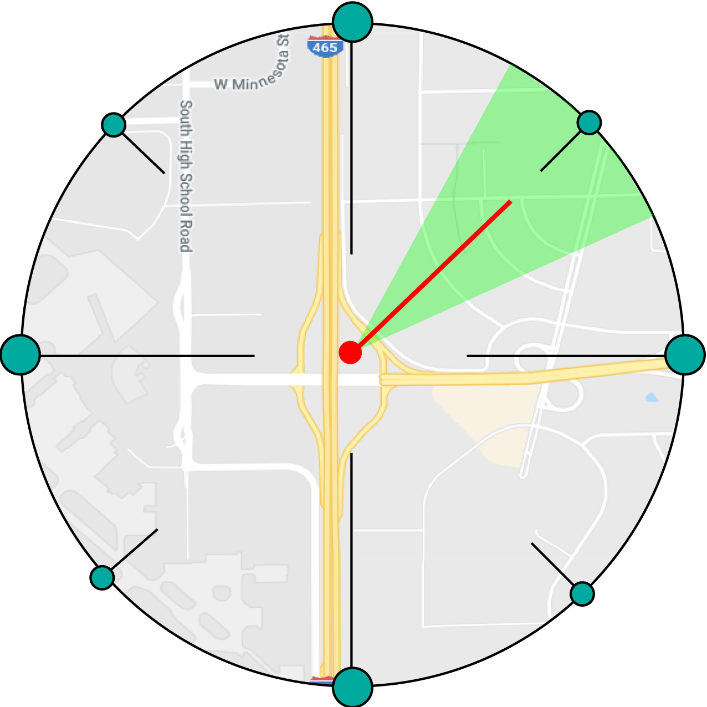


Figure 5

Description

* The purpose of this compass is to offer the operator a reference as to where the camera is currently pointing to. It gives a clear view of where it is pointing (red lines direction), the area it is currently viewing (green cone around the red line), and the current zoom level (the length of red line).

Pros:

* Easy to understand
* Strong references
* Can Control Camera
* Contains 8 preset locations for viewing
* Shows small area map

Cons:

* May require a calibration to ensure proper references
* If needed, a set calibration point and features will have to be added to each camera, resulting in labor costs

Subsystem 2

Incident List (Camera specific)  
Description

* The camera specific Incident List is a dumbed down version of the master one. It provides the operator access to incidents for that specific camera as well as resolved ones. It also gives the operator what camera it is and its location.

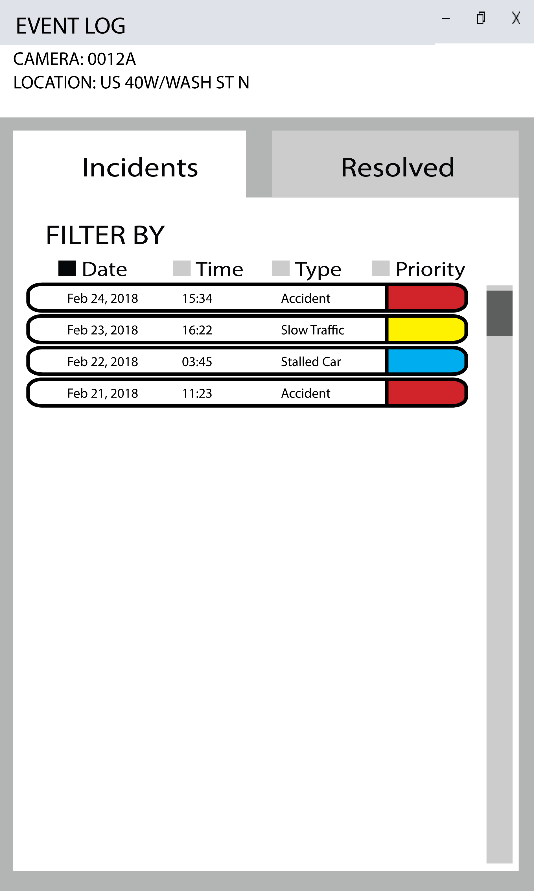


Figure 6

Pros:

* Gives camera specific information
* Contains same functionality as master incident list
* Allows Operator to go through each camera individually and spot incidents per camera
* Allows Operator to look through camera Specific incidents

Cons:

* Could have issues with getting data quickly and accurately
* May have issues with Operators not seeing Recent Incidents

Subsystem 3

Full Screen Video Feed

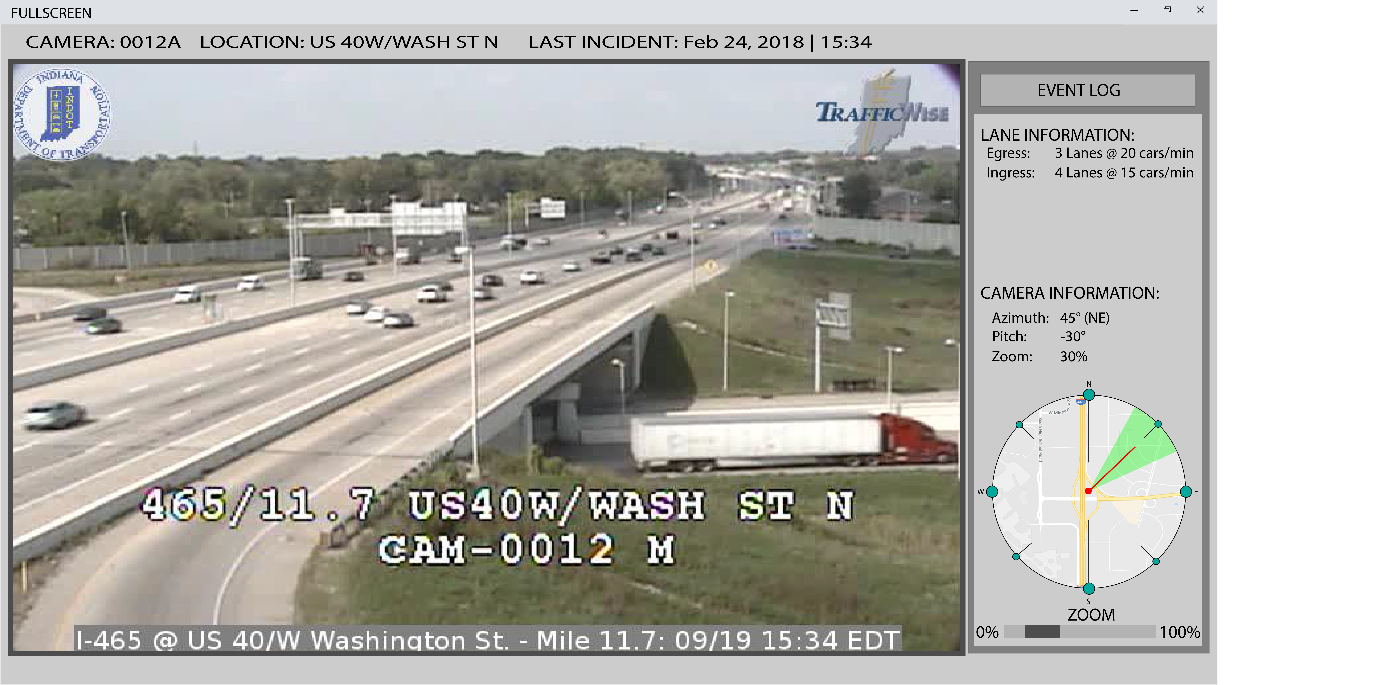
****

Figure 7

Description

* Upon user clicking full screen mode for camera, the above screen will pop up. It will have a larger video size, to view incidents with ease. It will also have clear information located all around the video box. Lastly it has the camera specific information along with the compass controls to easily locate and solve traffic issues.

Pros:

* Large Screen Video for easier viewing
* Camera specific information to allow user to know which camera they are located at.
* Camera Compass for location reference as well as control.
* It also has access to the camera specific event log

Cons:

* Could have issues with getting data quickly and accurately
* May have issues with Operators not seeing Recent Incidents
* May require a calibration to ensure proper references
* If needed, a set calibration point and features will have to be added to each camera resulting in labor costs

Software Information Diagram

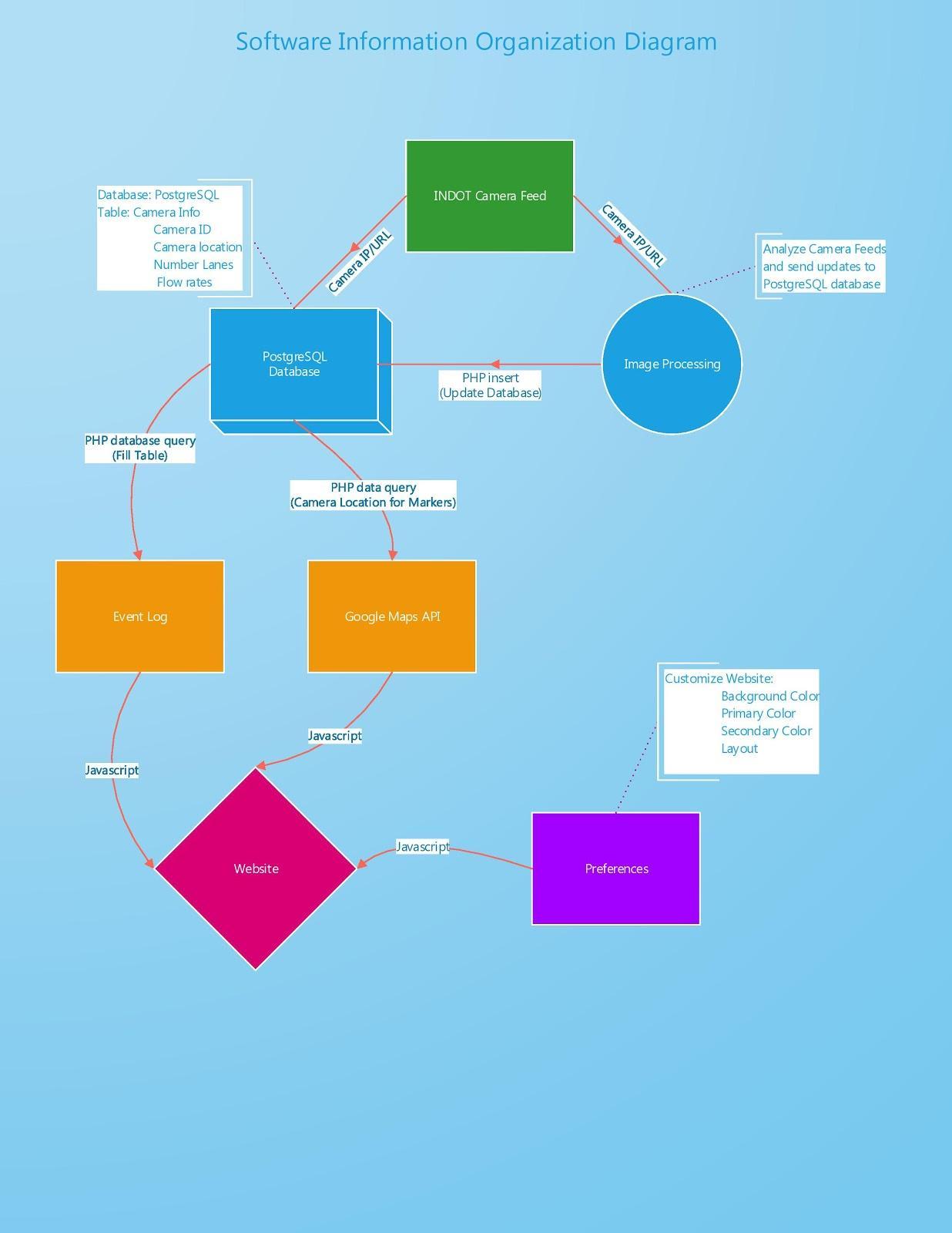


Figure 8

Database block diagram

The database block diagram is use to show how the database is implemented in the graphic user interface (GUI).

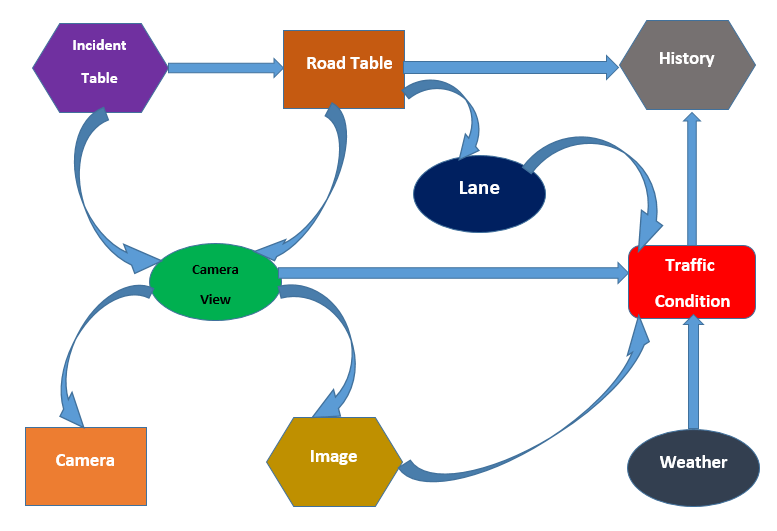
****

Figure 9

A database is any collection of data or information that is specially organized for rapid search and retrieval by a computer. In the figure 9 above, we can see that the incident table, the road table, the camera and the image are all linked to the camera view which is the most important key in the database. The image, the weather and the lane is linked to the traffic condition which is also linked to the history table.

Final Database Block Diagram from MySQL Workbench

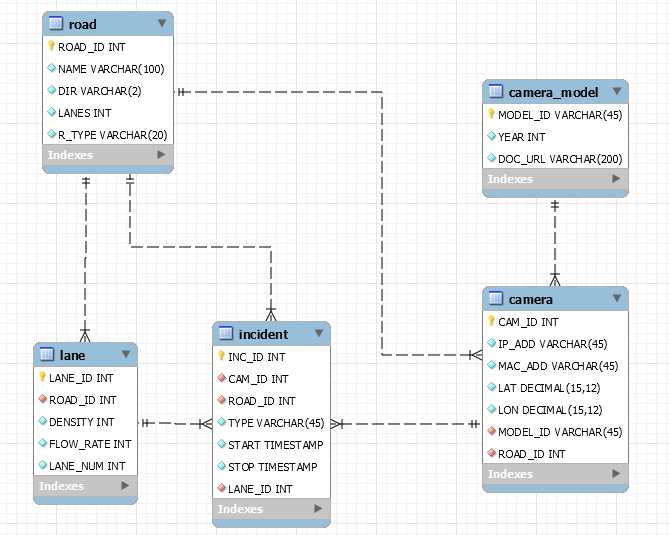


Figure 10

Implementation Details

Past Schedule of Implementation

In September, the first phase of the project consisted of designing the GUI. Successive GUI design iterations building off of the past iteration can be found in figures 1-3. No Design Options were applicable here because each successive iteration was reviewed by Dr. Chien and INDOT and decided what they liked and did not like about each iteration.

In October, development of the GUI began in order to keep track with the original requested schedule that a prototype be produced by the end of December. At the end of October, the leaflet map was chosen instead of google maps based on cost, and the map element was successfully implemented on the website. Table elements for the incident and camera list explorer were put in place with intent to add further functionality at a later date.

Furthermore, at the end of November sort-by functionality was introduced to the incident explorer table. This function was primitive at first but was later expanded in the month of March when test incidents were introduced. Initially it only sort by camera ID, road ID and GPS coordinates. GPS coordinates were obtained and used to label the icons on the map in their correct locations. However, the camera information provided was incomplete as a considerable portion of list did not include coordinates. For security reasons the INDOT camera feeds were not accessible outside of their private network. Consequently, public IP cameras of highways were used for video feeds for testing.

Moreover, some features initially requested by TASI had to be removed after a feature review meeting with INDOT due to security concerns, many of these removed features were relating to write access of the database. The first removed features was the writable notes in the info box, so that operators could write descriptions for each incident that could be pushed to the database and be viewed later. Another removed feature was control of the video cameras from the website, in addition to their joystick control. This included video camera directional angle presets to allow fast camera orientation. The last initial feature cuts was an administration page, which provided login and account management control. INDOT expressed that these permission features would be handled internally.

The work on the database began at the end of November. At first, the database only included basic tables such as the cameras and corresponding GPS coordinates. This database, as well as the apache server, was hosted on a raspberry pi until February due to ease of testing and full permissions access. TASI desired access to the test servers which prompted a change of hosting to IU’s network. Unfortunately utilizing the IU network meant a heavy reduction in user permissions for the database and slowed down development. The design of the database changed frequently but was initially created in October. The final iteration was settled on in February. Figure 9 shows an early design of the database that was produced in January and Figure 10 shows the final database from MySQL workbench. UITS tier two support was contacted in an effort to heighten development permissions of the database. Initial cooperation with UITS tech support did not provide an immediate solution, but eventually lead to another contact within UITS which was able to help with some of the critical permission bottlenecks that were slowing our development.

Meanwhile, the image processing team was working on traffic detection. Progress was focused on detecting individual vehicles and deriving a flow rate and traffic density. Some issues that the image processing had was detecting traffic during low light and night situations. Despite creating successful image processing software, no data was able to be provided to the database which impeded testing operations. To work around this issue, the database was populated with test entries to the incident list.  Each of these incidents had varying degrees of priority and location. The number of the priority flag gave rise to the color coding of the severity. Some sample test incidents are provided in figure 11. Intuitively, the severity of incidents increases in order from the shade of green to red. Red is the most severe incident with green being totally normal road with no current incidents.

**Implementation Results**

The following three figures shows the finalized GUI. The first shows the incident explorer focused on the static camera information which is pulled from the database. The second highlights the ability to enlarge the current video camera feed and to replace the main incident explorer with the text info box. The third shows the incident explorer focused on the dynamic incident information which is only test incidents at this point. This is also pulled from the database.

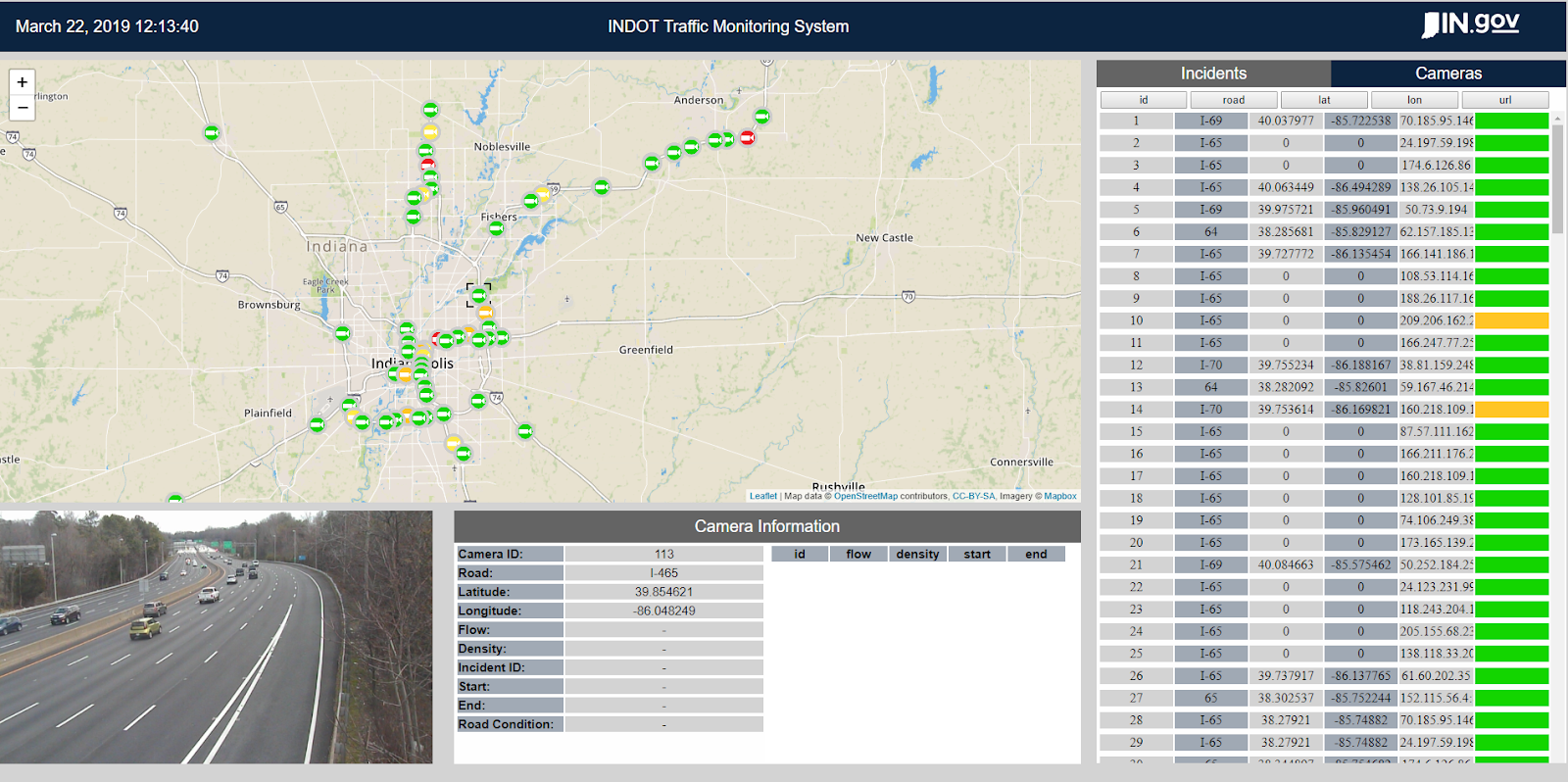
****

Figure 11

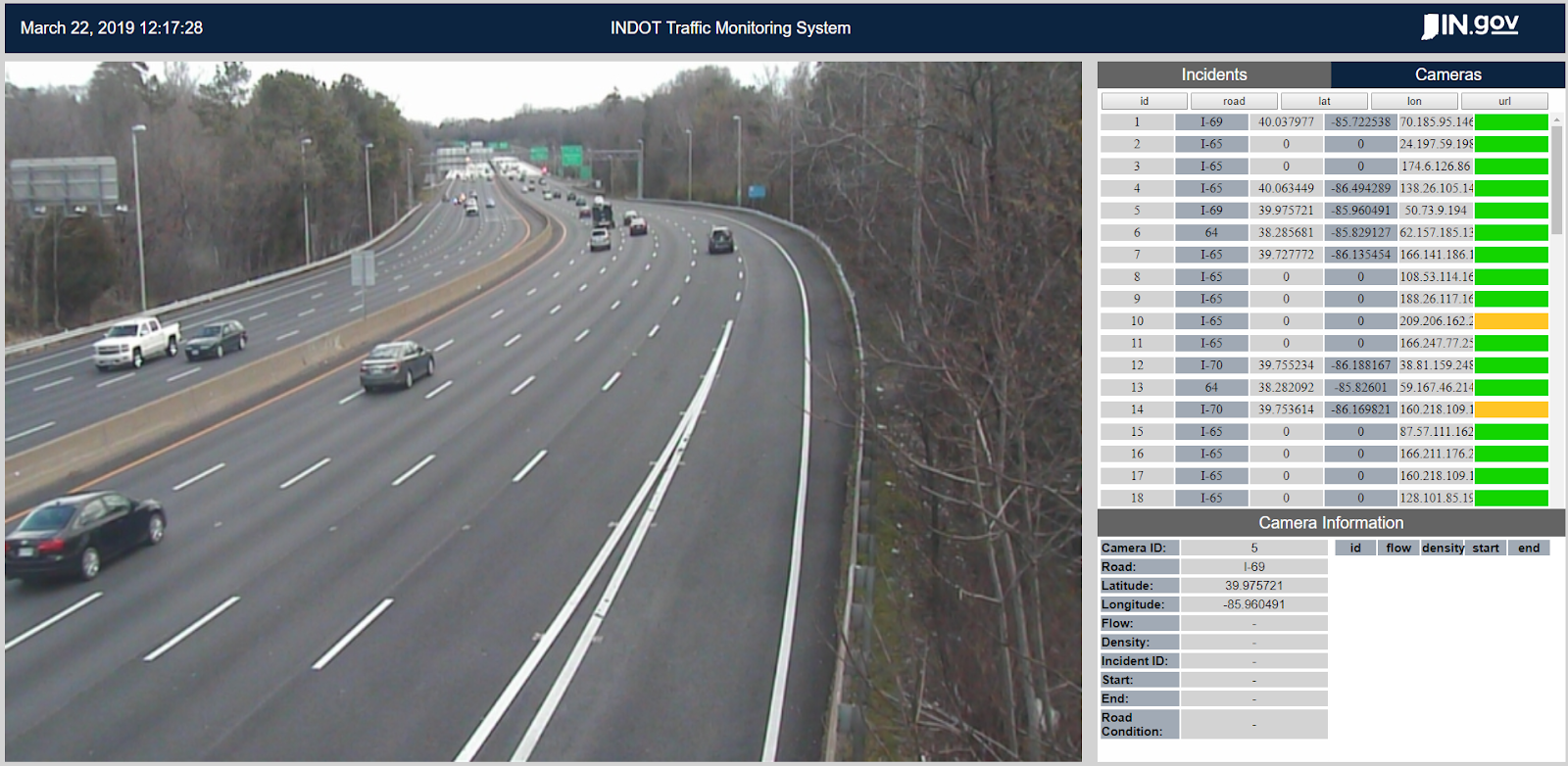
****

Figure 12

****

Figure 13

Remaining Schedule of Implementation

All that remains is shown in the project plan in Figure 14 below. This includes making the GUI more pleasing to the eye, adding video feed IP addresses, adding video camera GPS coordinates to the map api and adding traffic incident data when it becomes available. Although much of the testing has already begun, this will continue as well. This is described in the following section.

**Project Plan**

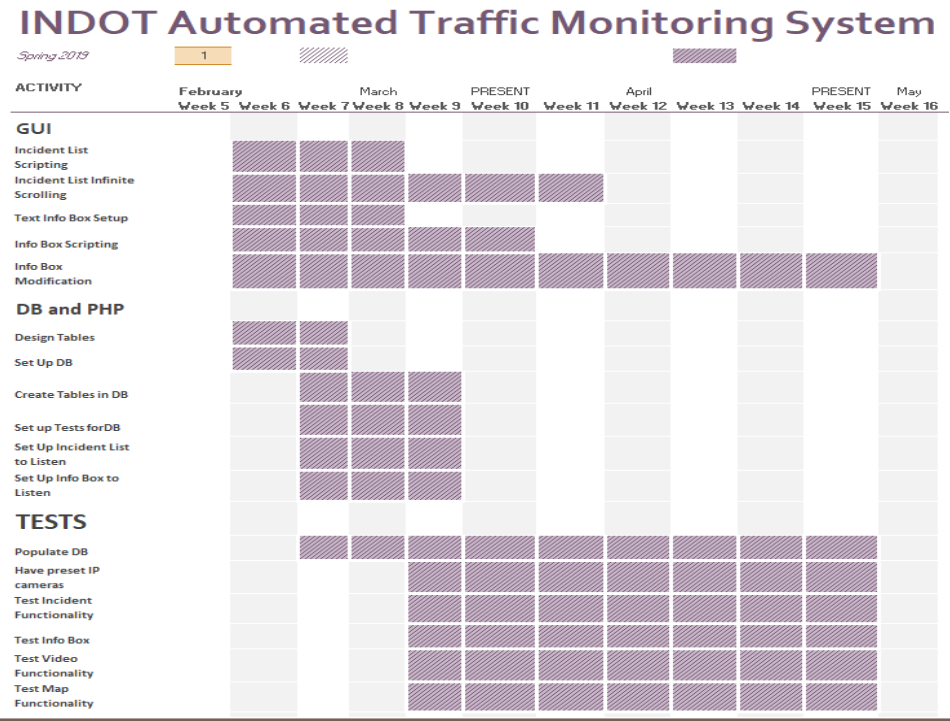


Figure 14

**Testing Procedures**

First, retrieve incidents from the database for just one camera. Then add more cameras to the database and retrieve their information as well. For the single camera test, create an incident in the database. See if this is both reflected on an overlaid camera icon on the google maps API and if this incident is reflected on the explorer list. Change the severity of the incident and check to see if both the respective camera icon and its entry in the explorer list change colors. Check to see that the text info box reflects this incident and also changes dynamically. When double clicking the camera icon, check to make sure that the full screen window of the camera feed for that camera enlarges. Make sure that it is up to date. If the user tries to open a camera feed while another camera feed is open, check to make sure that the first camera feed is closed out or overwritten.

Repeat these tests for more cameras until all 350 are linked. Check to see that the website handles 3-8 Mb/s with just one camera streaming in conjunction with the constantly refreshing google map API. Check to see that this happens simultaneously with dynamically colored camera icons and explorer incidents that are constantly changing. Check to make sure that the website doesn’t crash if the user tries to load every incident since the beginning of time either in the main explorer or in the text info box’s specific camera history explorer

Testing Results and Analysis

The GUI and database pass the tests described above under the current circumstances. With just a few test incidents, everything functions as it was intended to. It remains to be seen how this system would handle interacting with the entire network of 350 cameras and their video feeds. It is also unclear if the system can handle querying from the database to load all incidents from the beginning of time.

**Conclusions and Future Work**

INDOT now has a mostly functional database and graphical user interface that stores traffic information in an organized manner. However, this project is part of a larger one which involves image processing performed by a team of graduate students. Although the team of graduate students has been able to calculate traffic flow rates, they have not successfully been able to detect traffic incident information such as traffic jams, collisions, flash floods, debris in the road, or ongoing incidents such as construction. The graduate team has also had only partial success with detecting vehicle types. So far they have been able to roughly estimate vehicle types, but they have not been able to detect specific license plate numbers or specific vehicle information such as year make and model. They have also not yet detected emergency response vehicles or specific lane information. All of these incomplete aspects of the larger project are the future work.

**REFERENCES**

Agafonkin, V. (2017). Leaflet API reference. Retrieved March 22, 2019, from   https://leafletjs.com/reference-1.4.0.html

**Appendix: Source code**

index.php file

This is the index of our website and where the main structure is set up. The header of the website includes an image which is hard coded with coordinates and drawn via an svg element. Besides the structuring of the divs and some table elements, all of the js code is included as well. There is also a style.css file, but it is way too long to include here. For the complete source code, refer to the final zipped source code file.

<!DOCTYPE html>

<html>

<head>

<title>Test</title>

<link rel="stylesheet" href="style.css?v=1.1">

</head>

<body style = "background-color:lightgrey" onload="startUp(); startTime(); setInterval(onTimerElapsed, 10000);">

<div class="navbar">

<div id="txt" class="txt" style="color: white">

</div>

<div class="title">INDOT Traffic Monitoring System</div>

<div class="logo">

<svg data-name="IN.gov Logo" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 120 32.26"

height="32" width="120" class="logo"> <defs><style>.cls-1{fill:#fff;}</style></defs><title>IN.gov</title>

<path class="cls-1" d="M20.21.1,9.8,0A3.6,3.6,0,0,0,8.53.26L6.81,1A.91.91,0,0,1,6.45,1,

.86.86,0,0,1,6.15,1l-1-.45A.84.84,0,0,0,4.2.68a.88.88,0,0,0-.26.63L3.63,24a1.63,1.63,0,

0,1-.25.75L1,28a3.65,3.65,0,0,0-.53,1.18L0,31.28a.83.83,0,0,0,.13.7.73.73,0,0,0,.57.28,

1,1,0,0,0,.31,0l2.71-.83a1.3,1.3,0,0,1,.7.05L5.58,32a1.25,1.25,0,0,0,.51.12A1.37,1.37,0,

0,0,7,31.77l.61-.56a.82.82,0,0,1,.49-.16h.07l.9.18.19,0a1.2,1.2,0,0,0,1.11-.7l.28-.65a.

15.15,0,0,1,.17-.07l1.94.68a1.11,1.11,0,0,0,.34.06,1.18,1.18,0,0,0,1-.57l.32-.55a3.91,

3.91,0,0,1,.61-.75l1.57-1.41a1.59,1.59,0,0,0,.46-1.37L17,25.57h0a2.72,2.72,0,0,0,1.35,

0l1.88-.53a.82.82,0,0,0,.45-1.34L20.18,23a.08.08,0,0,1,0-.07.09.09,0,0,1,0-.06l.31-.

23a1.76,1.76,0,0,0,.64-1.28l.07-20.19A1,1,0,0,0,20.21.1Zm61.61,2H78.65A.61.61,0,0,0,

78,2.7a.61.61,0,0,1-1,.47,6.25,6.25,0,0,0-4.1-1.49c-4.19,0-7.31,3-7.31,8.44s3.16,8.44,

7.31,8.44A6.22,6.22,0,0,0,77,17a.61.61,0,0,1,1,.46v.16c0,3.26-2.47,4.19-4.56,4.19a6.79,

6.79,0,0,1-4.66-1.55.61.61,0,0,0-.93.14l-1.27,2.05a.62.62,0,0,0,.14.81,10.1,10.1,0,0,0,

6.72,2.12c4.08,0,9-1.54,9-7.86V2.7A.61.61,0,0,0,81.82,2.09ZM78,12.59a.63.63,0,0,1-.14.39,

4.9,4.9,0,0,1-3.6,1.7c-2.47,0-4.19-1.72-4.19-4.56s1.72-4.56,4.19-4.56a4.79,4.79,0,0,1,

3.63,1.73.59.59,0,0,1,.11.35Zm15.59,6.48a8.41,8.41,0,0,0,8.78-8.72,8.77,8.77,0,0,0-17.54,

0A8.4,8.4,0,0,0,93.63,19.07Zm0-13.52c2.71,0,4.26,2.23,4.26,4.8s-1.54,4.84-4.26,4.84S89.41,

13,89.41,10.36,90.92,5.55,93.63,5.55Zm25.49-3.47h-2.8a.88.88,0,0,0-.82.57l-3.29,8.78a.88.88,

0,0,1-1.64,0l-3.29-8.78a.88.88,0,0,0-.82-.57H103.7a.88.88,0,0,0-.81,1.2l6,14.82a.88.88,

0,0,0,.81.55h3.52a.88.88,0,0,0,.81-.55l6-14.82A.88.88,0,0,0,119.12,2.09ZM61.59,21.23a2,

2,0,1,0,2,2A2,2,0,0,0,61.59,21.23ZM30.18.44H26.66a.88.88,0,0,0-.88.88v23a.88.88,0,0,0,

.88.88h3.52a.88.88,0,0,0,.88-.88v-23A.88.88,0,0,0,30.18.44Zm25.47,0H52.13a.88.88,0,0,

0-.88.88V13.74a.88.88,0,0,1-1.59.51L40.05.81a.88.88,0,0,0-.71-.37H35.25a.88.88,0,0,

0-.88.88v23a.88.88,0,0,0,.88.88h3.52a.88.88,0,0,0,.88-.88v-13a.88.88,0,0,1,1.59-.51l9.95,

14a.88.88,0,0,0,.71.37h3.76a.88.88,0,0,0,.88-.88v-23A.88.88,0,0,0,55.65.44ZM118.59,

21.6H87.28a.88.88,0,0,0-.88.88v1.75a.88.88,0,0,0,.88.88h31.31a.88.88,0,0,0,.88-.88V22.

48A.88.88,0,0,0,118.59,21.6Z" transform="translate(0 0)"></path></svg>

</div>

</div>

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<div class="content">

<!--The following section is for the Incident List-->

<div class="vertical-menu" id="vmenu">

<div class="blockTop" style="width: 100%; overflow: hidden;">

<div style="width:50%; float: left; text-align: center;">

<button type="button" id="incidentButton" onclick="incidentButtonPressed()" class="block">Incidents</button>

</div>

<div style="width:50%; float: right; text-align: center">

<button type="button" id="cameraButton" onclick="cameraButtonPressed()" class="block">Cameras</button>

</div>

</div>

<div class="incidentTable" id="txtHint"></div>

</div>

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<div class="main" id="main">

<!--TThe following section is for the maps API-->

<div id="map" class="map" id="map">

</div>

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<div class="intel" id="intel">

<!--The following section is for the maps active camera-->

<div class="cam" id="thumbnail" >

<input id="liveFeed" type="image" onerror="imgError(this)"

src="http://87.57.111.162:81/mjpg/video.mjpg" onclick="toggleFocus()">

</div>

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<!--The following section is for the Info box-->

   <div class="infobox" id="infobox\_element" style="height: 100%; width: 58%;">

<div class="infobox-header">Camera Information</div>

<div style="height=100%; float=left;">

<table class="cbody" style="height=100%; width:50%;">

   <tbody ><tr>

   <th>Camera ID:</th>

   <td id="info\_cid">-</td>

   </tr>

   <tr>

   <th>Road:</th>

   <td id="info\_road">-</td>

   </tr>

   <tr>

   <th>Latitude:</th>

   <td id="info\_lat">-</td>

   </tr>

   <tr>

   <th>Longitude:</th>

   <td id="info\_lon">-</td>

   </tr>

   <tr>

   <th>Flow:</th>

   <td id="info\_flow">-</td>

   </tr>

   <tr>

   <th>Density:</th>

   <td id="info\_den">-</td>

   </tr>

   <tr>

   <th>Incident ID:</th>

   <td id="info\_iid">-</td>

   </tr>

   <tr>

   <th>Start:</th>

   <td id="info\_start">-</td>

   </tr>

   <tr>

   <th>End:</th>

   <td id="info\_end">-</td>

   </tr>

   <tr>

   <th>Road Condition:</th>

   <td id="info\_rc">-</td>

   </tr>

</tbody></table></div>

</div>

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<script src="leaflet.js"></script>

<script src="clock.js"></script>

<script src="map.js"></script>

<script src="resize.js"></script>

<script src="imgError.js"></script>

<script src="database.js"></script>

<script src="CameraInfoBox.js"></script>

</div>

</body>

</html>

**activeCam.php**

This is for clicking the camera icons, so that the URL is pulled by matching the latitude and longitude. Then it sends the URL to the videobox and displays the camera feed.

<?php

$lat = $\_GET["lat"];

$lon = $\_GET["lon"];

define('DB\_SERVER', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_USERNAME', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_PASSWORD', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_NAME', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

/\* Attempt to connect to MySQL database \*/

$link = mysqli\_connect(DB\_SERVER, DB\_USERNAME, DB\_PASSWORD, DB\_NAME);

$result = $link->query("SELECT url FROM test WHERE lat='".$lat."' AND lon='".$lon."'") or die("Bad Query: $sql");

while($row=mysqli\_fetch\_assoc($result)){

echo $row['url'];}

/\*"http://".$row['url']."/mjpg/video.mjpg";}\*/

mysqli\_close($link);

?>

**cameraIcons.php**

This places camera icons on the leaflet map using the leaflet API. It searches the database and places icons at every coordinate for each camera. Any cameras without specifically provided coordinates is still placed but it is located at the intersection of the equator and the prime meridian since its latitude and longitude coordinates remain 0,0.

<?php

define('DB\_SERVER', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_USERNAME', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_PASSWORD', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_NAME', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

/\* Attempt to connect to MySQL database \*/

$link = mysqli\_connect(DB\_SERVER, DB\_USERNAME, DB\_PASSWORD, DB\_NAME);

$result = $link->query("SELECT \* FROM test") or die("Bad Query: $link");

while($row=mysqli\_fetch\_assoc($result)){

echo "L.marker([".$row['lat'].",".$row['lon']."], {icon: greenIcon}).addTo(mymap).on('click', onClick);";

}

mysqli\_close($link);

?>

**cameraTable.php**

This pulls data from the database for camera information and populates the camera explorer.

<?php

define('DB\_SERVER', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_USERNAME', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_PASSWORD', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_NAME', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

/\* Attempt to connect to MySQL database \*/

$link = mysqli\_connect(DB\_SERVER, DB\_USERNAME, DB\_PASSWORD, DB\_NAME);

$sql = "SELECT \* from test";

$result = $link->query($sql) or die("Bad Query: $sql");

$data = array();

while($rows = mysqli\_fetch\_assoc($result)){

$data[] = $rows;}

echo json\_encode($data);

mysqli\_close($link);

?>

**incidentTable.php**

This places the data for the incident portion of the table.

<?php

define('DB\_SERVER', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_USERNAME', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_PASSWORD', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

define('DB\_NAME', '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

/\* Attempt to connect to MySQL database \*/

$link = mysqli\_connect(DB\_SERVER, DB\_USERNAME, DB\_PASSWORD, DB\_NAME);

$sql = "SELECT \* from test\_incident";

$result = $link->query($sql) or die("Bad Query: $sql");

$data = array();

while($rows = mysqli\_fetch\_assoc($result)){

$data[] = $rows;}

echo json\_encode($data);

mysqli\_close($link);

?>

**Javascript Functions**

**cameraInfoBox.js**

Upon clicking a camera table or incident table row it pulls the relevant information out and places it into the camera info box.

function camerainfobox(id){

var t\_name = id.split("\_");

var temp\_obj;

//if parsed value is from incident and is non-zero

if( ( t\_name[0] == "in" ) && ( incidentTable != 0 ) ){

temp\_obj = incidentTable[parseInt(t\_name[1]) - 1];

document.getElementById("info\_cid").innerHTML = temp\_obj.camid;

document.getElementById("info\_road").innerHTML = temp\_obj.road + " " + temp\_obj.milemarker;

document.getElementById("info\_lat").innerHTML = "-";

document.getElementById("info\_lon").innerHTML = "-";

document.getElementById("info\_flow").innerHTML = temp\_obj.flow\_rate;

document.getElementById("info\_den").innerHTML = temp\_obj.density;

document.getElementById("info\_iid").innerHTML = temp\_obj.id;

document.getElementById("info\_start").innerHTML = temp\_obj.start\_time;

document.getElementById("info\_end").innerHTML = temp\_obj.end\_time;

document.getElementById("info\_rc").innerHTML = "-";

}

//if parsed value is from camera table then find camera from array

if( ( t\_name[0] == "cam" ) && ( cameraTable != 0 )){

for(i=0; i<cameraTable.length ; i++){

temp\_obj = cameraTable[i];

if( temp\_obj.id == t\_name[1] ){

i = cameraTable.lenghth + 1;

}

}

document.getElementById("info\_cid").innerHTML = temp\_obj.id;

document.getElementById("info\_road").innerHTML = temp\_obj.road;

document.getElementById("info\_lat").innerHTML = temp\_obj.lat;

document.getElementById("info\_lon").innerHTML = temp\_obj.lon;

document.getElementById("info\_flow").innerHTML = "-";

document.getElementById("info\_den").innerHTML = "-";

document.getElementById("info\_iid").innerHTML = "-";

document.getElementById("info\_start").innerHTML = "-";

document.getElementById("info\_end").innerHTML = "-";

document.getElementById("info\_rc").innerHTML = "-";

console.log("trying to find:" + temp\_obj.url);

updateLiveFeed(temp\_obj.url);

}

}

function iconinfobox(lat, lon){

var temp\_obj;

for(i=0; i<cameraTable.length ; i++){

temp\_obj = cameraTable[i];

if( ( temp\_obj.lat == lat.toString() ) && ( temp\_obj.lon == lon.toString() ) ){

i = cameraTable.lenghth + 1;

}

}

document.getElementById("info\_cid").innerHTML = temp\_obj.id;

document.getElementById("info\_road").innerHTML = temp\_obj.road;

document.getElementById("info\_lat").innerHTML = temp\_obj.lat;

document.getElementById("info\_lon").innerHTML = temp\_obj.lon;

document.getElementById("info\_flow").innerHTML = "-";

document.getElementById("info\_den").innerHTML = "-";

document.getElementById("info\_iid").innerHTML = "-";

document.getElementById("info\_start").innerHTML = "-";

document.getElementById("info\_end").innerHTML = "-";

document.getElementById("info\_rc").innerHTML = "-";

updateLiveFeed(temp\_obj.url);}

**clock.js**

This creates the current date with the active time to use in the website.

function startTime() {

  var today = new Date();

var months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"];

var h = today.getHours();

  var m = today.getMinutes();

  var s = today.getSeconds();

  m = checkTime(m);

  s = checkTime(s);

  document.getElementById('txt').innerHTML =

  months[today.getMonth()] + " " + today.getDate() + ", " + today.getFullYear() + " " + h + ":" + m + ":" + s;

  var t = setTimeout(startTime, 500);

}

function checkTime(i) {

  if (i < 10) {i = "0" + i};  // add zero in front of numbers < 10

  return i;

**}**

**database.js**

This creates sorting elements as well as many other important functions. These functions run in the background and are interfaced with other functions.

var currentTable = 0; //keeps track of current table

var incidentTable = 0;

var cameraTable = 0;

var scrollHeight = 0;

function updateLiveFeed(url) { //sets url for live camera; takes: url, returns: nothing

   document.getElementById("liveFeed").src = "http://"+url+"/mjpg/video.mjpg";

}

//functions

function kyleSort(property, table) { //sort javascript object by key;  takes: key, returns: nothing

   var tempTable;

var tempRowA, tempRowB;

var i, j;

if(table=="1"){

tempTable = cameraTable;

}else if(table=="2"){

tempTable = incidentTable;

}

for(i=0;i<tempTable.length - 1;i++){

for(j=0;j<tempTable.length - 1;j++){

tempRowA = tempTable[j];

tempRowB = tempTable[j+1];

if(kyleCompare(tempRowA[property],tempRowB[property])>0){

tempTable[j] = tempRowB;

tempTable[j+1] = tempRowA;

}

}

}

}

function kyleCompare(a, b){ //returns smaller or equal

var tempA, tempB;

if(isNaN(a)){

tempA = a.toUpperCase();

tempB = b.toUpperCase();

//return 1 if b comes first, -1 if a comes first, 0 if same, case doesnt matter

return tempA.localCompare(tempB);

}

else{

if(parseFloat(a)<parseFloat(b)){

return -1;

}

else if(a>b){

return 1;

}

else if(a==b){

return 0;

}

}

}

//makeTable

function makeTable(obj){ //turn javascript object into html table;  takes: javascript Object, returns: string

var table = "<table class='fixed\_header' id='vmenu\_table'>";

var header = Object.keys(obj[0]);

var tempSortType = 0;

var row = new Array(Object.keys(obj[0]).length);

table += "<thead>";

table += "<tr>";

for(i=0;i< Object.keys(obj[0]).length; i++){

tempSortType = "'"+header[i]+"'";

table += "<th><form><input type='button' value='"+header[i]+"' id='id\_"+header[i]+"' onclick='sortTable(\""+header[i]+"\")' style='width: 100%'></form></th>";

}

table += "</tr></thead><tbody id='tbody\_id'>";

for(i=0;i<obj.length;i++){

row = Object.values(obj[i]);

if(localStorage.getItem("activeList")==1){  //This looks to see if cameras is active

table += "<tr input type='button' onclick='camerainfobox(\"cam\_"+row[0]+"\")'>";

}

if(localStorage.getItem("activeList")==2){ //This looks to see if incident is active

table += "<tr input type='button' onclick='camerainfobox(\"in\_"+row[0]+"\")'>";

}

for(j=0;j<Object.keys(obj[0]).length; j++){

table += "<td>";

table += row[j];

table += "</td>";

}

table += "<td id='cell\_"+row[0]+"'></td>";

table += "</tr>";

}

table += "</tbody></table>";

return table;

}

//setCurrentTable

function tableColor(){

if(localStorage.getItem("activeList")==1){  //This looks to see if cameras is active

var x;

var flag = 0;

var temp;

var inTmp;

var camTmp;

for(i=0;i<cameraTable.length;i++){

x = i+1;

temp = 'cell\_'+x;

camTemp = cameraTable[i];

flag = 0;

for(j=0;j<incidentTable.length;j++){

inTemp = incidentTable[j];

if(inTemp.camid == camTemp.id){

document.getElementById(temp).style.backgroundColor = "red";

j=incidentTable.length+5;

flag = 0;

}

}

if(j==incidentTable.length){

document.getElementById(temp).style.backgroundColor = "lightgreen";

}

}

}

if(localStorage.getItem("activeList")==2){ //This looks to see if incident is active

for(i=0;i<incidentTable.length;i++){

x = i+1;

temp = 'cell\_'+x;

document.getElementById(temp).style.backgroundColor = "lightgreen";

}

}

}

function setCameraTable(curTable){ //sets current active table;  takes: json, returns: nothing

window.cameraTable = curTable;

if(localStorage.getItem('activeList')=='1'){

setCurrentTable();

}

}

function setIncidentTable(curTable){ //sets current active table;  takes: json, returns: nothing

window.incidentTable = curTable;

if(localStorage.getItem('activeList')=='2'){

setCurrentTable();

}

}

function cameraButtonPressed(){

activateCameraTable();

selectorGroup.clearLayers();

}

function incidentButtonPressed(){

activateIncidentTable();

selectorGroup.clearLayers();

}

function setCurrentTable(){ //sets current active table;  takes: json, returns: nothing

if(localStorage.getItem('activeList')=='1'){ //camera

kyleSort(localStorage.getItem('sortType'),localStorage.getItem('activeList'));

scrollHeight = document.querySelector('tbody').scrollTop;

document.getElementById("txtHint").innerHTML = makeTable(window.cameraTable);

document.querySelector('tbody').scrollTop = scrollHeight;

createCameraIcons();

tableColor();

}

if(localStorage.getItem('activeList')=='2'){ //incident

kyleSort(localStorage.getItem('sortType'),localStorage.getItem('activeList'));

scrollHeight = document.querySelector('tbody').scrollTop;

document.getElementById("txtHint").innerHTML = makeTable(window.incidentTable);

document.querySelector('tbody').scrollTop = scrollHeight;

createIncidentIcons();

tableColor();

}

}

function createCameraIcons(){//<?php include 'cameraIcons.php';?>

markerGroup.clearLayers();

var camTmp;

var inTmp;

var tempMarker = 0;

for(i=0;i<cameraTable.length;i++){

camTmp = cameraTable[i];

for(j=0;j<incidentTable.length;j++){

inTmp = incidentTable[j];

if(camTmp.id == inTmp.camid){

tempMarker = L.marker([camTmp.lat,camTmp.lon], {icon: orangeIcon}).on('click', onClick);

j=incidentTable.length+5;

}

}

if(j==incidentTable.length){

tempMarker = L.marker([camTmp.lat,camTmp.lon], {icon: greenIcon}).on('click', onClick);

}

markerGroup.addLayer(tempMarker);

}

markerGroup.addTo(mymap);

}

function createIncidentIcons(){//<?php include 'cameraIcons.php';?>

markerGroup.clearLayers();

var inTemp = 0;

var camTemp = 0;

var tempMarker = 0;

for(i=0;i<incidentTable.length;i++){

inTemp = incidentTable[i];

for(j=0;j<cameraTable.length;j++){

camTemp = cameraTable[j];

if(inTemp.camid == camTemp.id){

tempMarker = L.marker([camTemp.lat,camTemp.lon], {icon: orangeIcon}).on('click', onClick);

markerGroup.addLayer(tempMarker);

j=cameraTable.length;

}

}

}

markerGroup.addTo(mymap);

}

//startUp

function startUp() {//startup settings; takes: nothing, returns: nothing

localStorage.setItem('sortType', 'id');

localStorage.setItem('activeList', '2');

getCameraTable();

getIncidentTable();

document.getElementById("incidentButton").style.backgroundColor="#0c2440";

}

//activateCameraTable

function activateCameraTable(){//activate camera table; takes: nothing, returns: nothing

localStorage.setItem('activeList', '1');

document.getElementById("incidentButton").style.backgroundColor="#646464";

document.getElementById("cameraButton").style.backgroundColor="#0c2440";

getCameraTable();

getIncidentTable();

}

//activateIncidentTable

function activateIncidentTable(){//activate incident table; takes: nothing, returns: nothing

localStorage.setItem('activeList', '2');

document.getElementById("cameraButton").style.backgroundColor="#646464";

document.getElementById("incidentButton").style.backgroundColor="#0c2440";

getCameraTable();

getIncidentTable();

}

//getActiveCamLL

function getActiveCamLL(lat,lon) {//get active camera settings; takes: lat, lon, returns: nothing (response is given to another function)

   if (window.XMLHttpRequest) {

       // code for IE7+, Firefox, Chrome, Opera, Safari

       xmlhttp = new XMLHttpRequest();

   } else {

       // code for IE6, IE5

       xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

   }

   xmlhttp.onreadystatechange = function() {

   if (this.readyState == 4 && this.status == 200) {

updateLiveFeed(this.responseText);

}

};

xmlhttp.open("GET","activeCam.php?lat="+lat+"&lon="+lon,true);

xmlhttp.send();

}

//getActiveCamID

function getActiveCamID(id) {//get active camera settings; takes: lat, lon, returns: nothing (response is given to another function)

   if (window.XMLHttpRequest) {

       // code for IE7+, Firefox, Chrome, Opera, Safari

       xmlhttp = new XMLHttpRequest();

   } else {

       // code for IE6, IE5

       xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

   }

   xmlhttp.onreadystatechange = function() {

   if (this.readyState == 4 && this.status == 200) {

updateLiveFeed(this.responseText);

}

};

xmlhttp.open("GET","activeCam.php?id="+id,true);

xmlhttp.send();

}

//getTable

function getIncidentTable() {//get active camera settings; takes: table name, returns: nothing (response is given to another function)

   if (window.XMLHttpRequest) {

       // code for IE7+, Firefox, Chrome, Opera, Safari

       xmlhttp = new XMLHttpRequest();

   } else {

       // code for IE6, IE5

       xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

   }

   xmlhttp.onreadystatechange = function() {

   if (this.readyState == 4 && this.status == 200) {

return setIncidentTable(JSON.parse(this.responseText));

}

};

xmlhttp.open("GET","incidentTable.php",true);

xmlhttp.send();

}

//getTable

function getCameraTable() {//get active camera settings; takes: table name, returns: nothing (response is given to another function)

   if (window.XMLHttpRequest) {

       // code for IE7+, Firefox, Chrome, Opera, Safari

       xmlhttp = new XMLHttpRequest();

   } else {

       // code for IE6, IE5

       xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

   }

   xmlhttp.onreadystatechange = function() {

   if (this.readyState == 4 && this.status == 200) {

return setCameraTable(JSON.parse(this.responseText));

}

};

xmlhttp.open("GET","cameraTable.php",true);

xmlhttp.send();

}

//onTimerElapsed

function onTimerElapsed(){//get active table for timer; takes: nothing, returns: nothing

if(localStorage.getItem('activeList')=='1'){

activateCameraTable();

} else{

activateIncidentTable();

}

}

//sortTable

function sortTable(name) {//sort javascript Object; takes: javascript Object, returns: nothing

localStorage.setItem('sortType', name);

if(localStorage.getItem('activeList')=='1'){

activateCameraTable();

}

else{

activateIncidentTable();}

}

**imgError.js**

If there's an issue with the URL a default picture pops up alerting that this issue with the URL.

document.getElementById("liveFeed").addEventListener("error", imgError);

function imgError(image) {

image.src = "CameraUnavailableError.png";

return true;

**leaflet.js**

Imported from leaflet API. See references section.

**map.js**

This is responsible for color coding the icons on the map based on their incident priority. It also handles the click event for icons so that the selection box indicator is placed over the active camera icon.

var mymap = L.map('map').setView([39.8, -86.157877], 11);

var markerGroup = L.layerGroup().addTo(mymap);

var selectorGroup = L.layerGroup().addTo(mymap);

L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access\_token=pk.eyJ1IjoibWFwYm94IiwiYSI6ImNpejY4NXVycTA2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLbB6B5aw', {

maxZoom: 18,

attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, ' +

'<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, ' +

'Imagery © <a href="https://www.mapbox.com/">Mapbox</a>',

id: 'mapbox.streets'

}).addTo(mymap);

var greenIcon = L.icon({

iconUrl: 'greenIcon.png',

iconSize:     [25, 25], // size of the icon

})

var orangeIcon = L.icon({

iconUrl: 'orangeIcon.png',

iconSize:     [25, 25], // size of the icon

});

var yellowIcon = L.icon({

iconUrl: 'yellowIcon.png',

iconSize:     [25, 25], // size of the icon

});

var activeMarker = L.icon({

iconUrl: 'selector.png',

iconSize: [30,30],

});

function onClick(e){

if(selectorGroup!=undefined){

selectorGroup.clearLayers();

}

var temp = L.marker(e.latlng, {icon: activeMarker});

selectorGroup.addLayer(temp);

getActiveCamLL(e.latlng.lat,e.latlng.lng);

iconinfobox(e.latlng.lat,e.latlng.lng);

}

**resize.js**

This is responsible for switching the camera thumbnail to full screen and back when clicked. It also handles the moving of the camera info box so that it remains visible on the right hand side of the screen.

var big = 0;

function toggleFocus(){

var feed = document.getElementById("thumbnail");

var info = document.getElementById("infobox\_element");

var map = document.getElementById("map");

var main = document.getElementById("main");

var intel = document.getElementById("intel");

var iList = document.getElementById("txtHint");

var vmenu = document.getElementById("vmenu");

console.log("toggle: " + big);

if(!big){

feed.style.position = "absolute";

feed.style.width="100%";

map.style.display = "none";

main.appendChild(feed);

vmenu.appendChild(info);

iList.style.height = "60%";

info.style.height = "40%";

info.style.width = "100%";

big = 1;

}else{

feed.style.position = "relative";

feed.style.width="40%";

map.style.display = "inline";

intel.appendChild(feed);

intel.appendChild(info);

iList.style.height = "100%";

info.style.height = "100%";

info.style.width = "58%";

big = 0;

}

}