Current Deliverable Milestone:

Final Project Notebook

Project:

Mountain Lion Detection System

Team:

Team Mountaineers

Due Date of Current Milestone:

8 Dec 2022

Executive Summary

Following a CS532 assignment briefing about the necessity of a ***'Mountain Lion Detection System'*** *for California Park Rangers, our group communally designed a reusable software system. We then went on to implement the system using a modern* ***web stack****. We utilized industry grade tools, like;* ***React.js, MongoDB & Node.js****, whilst following all the proper agile design philosophies. Many commits later and the group was able to deliver a fully functional prototype that can effectively support rangers to classify animal detections.*

Contents Page

Signature Page

Lewis Gibney (PM): Lewis Gibney Date: 22 Sept 2022

Halah Salman: Halah D. Salman Date: 22 Sept 2022

June Vincent Magat: June Magat Date: 22 Sept 2022

Project Plan

Project Team

The team, “The Life Comrades”, currently has 3 team members following a group fusion and people dropping the class. Each member has specific roles;

- Lewis Gibney - Project Manager, UI Designer - Incharge of making sure the team works cohesively, whilst also utilizing past experience to implement an industry grade UI.
- Halah Salman - Test Manager, Programmer - Responsible for assuring proper testing through each of the different components and also aiding in implementing the system design.
- June Vincent Magat - Chief Programmer - Incharge of implementing the system design in a robust, reusable and reliable manner.

Environment

After digesting the project, it was understood that it roughly followed a CRUD application structure. Applications that have a requirement to manage persistent data often use DBs (databases). DBs are a key technology within the web-sphere. Therefore, due to the necessity of a DB within the project and integrity of the technology within the web, we are going to use a modern and up-to-date web stack to implement the project. This means the main codebase of the project will be generated using HTML, CSS and JavaScript.

Furthermore, we will be using React as it provides an open source well maintained codebase that allows us to get the most from a web application. On top of react, we will use Bootstrap to easily manage the UI elements whilst also achieving a modern aesthetic. We will require a DB to store the data, at this preliminary stage we plan to use MongoDB self hosted as it’s a very popular tried-and-tested platform.

The project will also require us to detect specific animal noises from detection sensors, for this we will use the Raspberry Pi platform coupled with python for our programming needs. Raspberry Pis have been chosen due to their relative cheapness and impressive functionality.

ICSM Common Case

The ICSM is a risk driven framework or meta-model that is built on principles and supports the creation of life-cycle processes based on the features, limitations, and risks associated with a specific project or program. The ICSM common cases have been developed to show the users how to use the framework. Because the goal of the ICSM is to identify the risks in software development these common cases are essentially situations that present risks at the various stages of the development. The common cases we will be considering for our project based on our project are:

Project	Cases	About the Component	It’s Use	Rationale
Mountain Lion Detection Application	Software application or system	Database management system, command and control/sensor processing system.	For our project we will have multiple databases to store the information, send and retrieve it.	-safety/security of critical components -for selecting and implementing solutions
Raspberry Pi	Software-intensive device	A piece of equipment that is developed for a special purpose and has significant features provided by the software.	For our Project we will be using Raspberry Pi and it’s Raspbian OS System to implement it.	-to implement the detection system and transfer data -meet certain price point -upgrade and fix problems
Multi Sensor, Laptop/s/phones (We aren’t developing the hardware this just shows all the components needed for the system)	Hardware platform	Hardware platform	A crucial part of the project requires the display to be able to use sensor/s to implement it/run it.	-to implement the system -to display the output of the system
Detection Management	System of System(SoS) or enterprise-wide system	The collection of systems we will be using to implement our project, which we can also update and fix.	For our project, we need to get data from the sensor and create a system that works together to recognize, work on, transfer and store the data in it.	-to integrate a set of existing systems -guide and evolve the integration of a set of existing systems
Maintenance	Brownfield modernization	Incremental replacement of old, fragile business systems with COTS products or technology refreshment/up-grading of existing systems.	For our project we will need to be able to fix and upgrade the system from time to time.	-old systems can contain bugs/vulnerabilities. -old systems need new features to keep it popular.

Methodologies

We will be using multiple of these methods which count as part of the ICSM(Incremental Commitment Spiral Model) Software Strategies to respond to the challenges that may arise from designing/implementing the Mountain Lion detection system.

Name	Priority level	Explanation
Architected Agile (Using Jira scrum agile methodology)	Low to Medium	We are using Architected Agile to create a strong software foundation and architecture, then transition to purely Agile Process.
Agile (Using Jira)	Low to Medium	We are using the Agile methods for the software development process to develop the software capabilities

Project Development Estimate

Software Size

Based on our lack of previous experience with writing and implementing large software systems, we expect our actual software size to vary from our estimates. However, with that said, we will try to use prior experience where applicable to estimate the size of the different components, those being; the detection program, the database management system & the user interface. We will focus on function point sizing.

Detection Program

The detection program, currently slated to be coded in python, will be of a rather average level of complexity with ~100 lines of code. The program will be incharge of reading sensor data from the raspberry PIs inputs, performing a calculation to check for a detection & finally returning a JSON holding detection details to the DB when the threshold is met.

Components	Low	Average	High	Details	Score
Internal Logic Files		1		One average python file	10
External Interface Files	1			Audio level details to trigger detection	5
External Inputs			1	Continuous audio signal	6
External Outputs			1	JSON with confirmation of detection	7
External Queries					0
				UFP Count	28

Database Management System

The DB will require code to manage the data as requested in the briefing. The system must be able to edit the DB based on time elapsed and update the DB with new incoming detections, whilst also being able to serve data when called for.

Components	Low	Average	High	Details	Score
Internal Logic Files			1	Code for delivering and managing saved data	15
External Interface Files					0
External Inputs			2	Database and detectors	12

External Outputs			1	Data for re-entry in to database	7
External Queries			1	Data for the user interface	6
				UFP Count	40

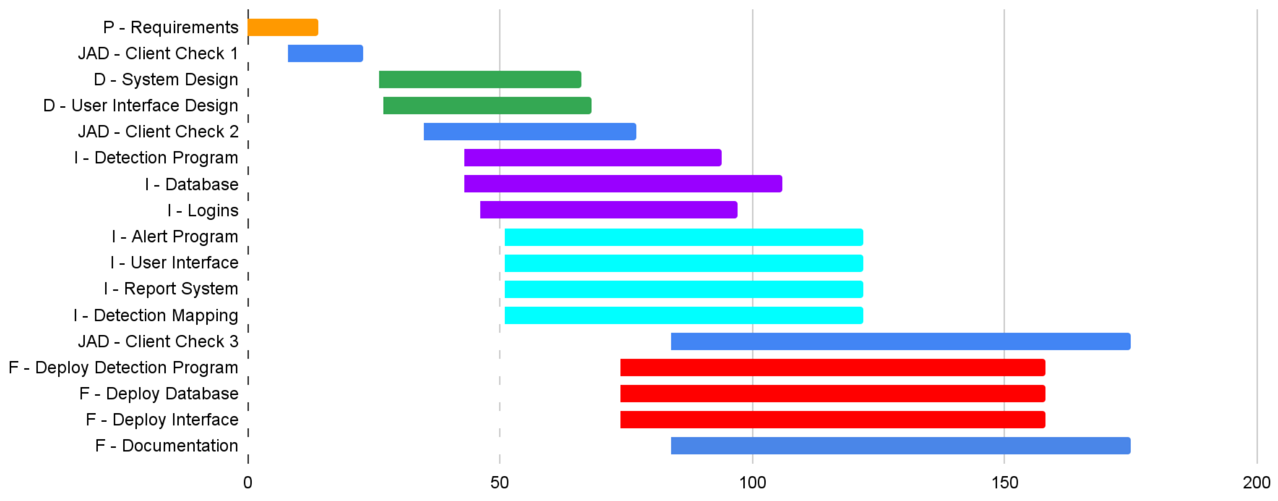
User Interface

The UI will be generated with react, so both our functional code and display code will be housed in the same files. At this early point of development we foresee 3 different pages; a landing/login page, a report page and a graphical report page (where the data is shown across a map). We estimate that the code for this section will be the largest.

Components	Low	Average	High	Details	Score
Internal Logic Files	0	1	2	3 different react page files	40
External Interface Files					0
External Inputs			1	Input from database management system	6
External Outputs					0
External Queries					0
				UFP Count	46

Schedule

We currently have the tasks scheduled to make full utilization of the semester. We start first in the preliminary phase where we take the time to break down the requirements and then start documentation. After completing the documents we go on to have our first client check. Following a positive or otherwise resolved client meeting, we head to the design phase. After designing the system we will have another meeting with the client to ensure they are happy with the initial design, following any alterations we start the implementation phase. The implementation phase has been split up into 2 sub-sections that can be roughly encapsulated as backend (detector, database and login system) and frontend (alert program, UI, report system and detection mapping). After completing the implementation phase we will have a prototype system working in a development environment, we will then show this to the client to ensure they are happy before the final stage. The final stage is to deploy the prototype to a physical hardware environment that is similar to its final form. In this final phase we will also confirm that all the proper documentation has been created.

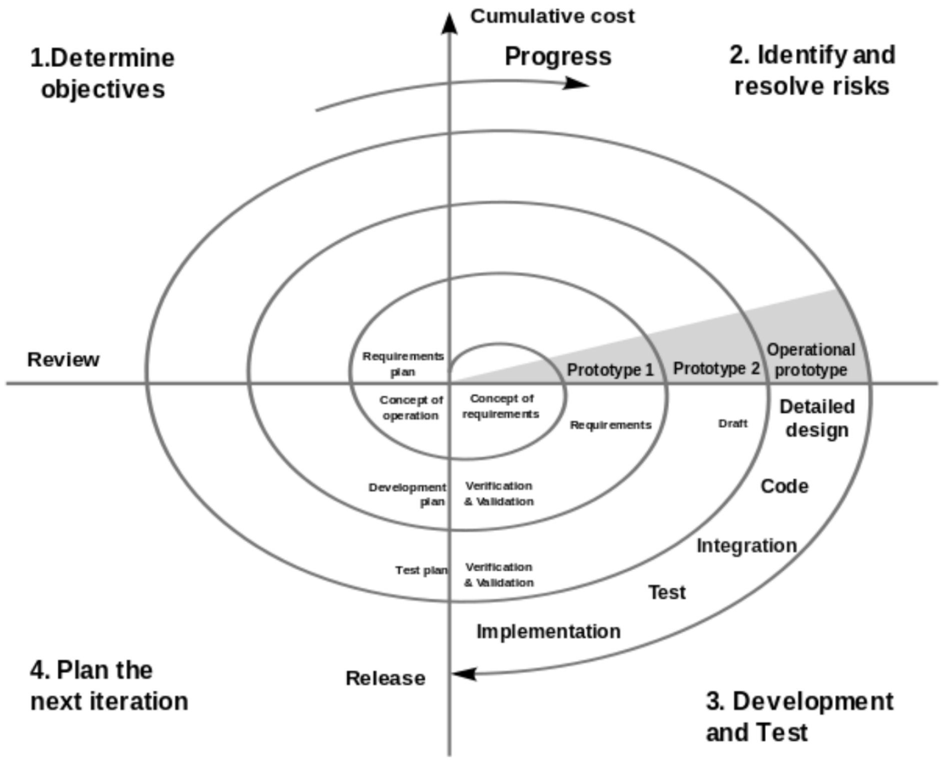


Total Estimated Hours to Execute Tasks

If we extrapolate the time from the schedule, by taking each day on each task as being an hour we can estimate the time the project will take to execute. This total is 213 hours, which should roughly end up being 71 hours per group member.

Life Cycle Model

The life cycle model we choose to go with is the Spiral model due to its unique ability to include a more detailed planning of the project. It is also more compatible with risk analysis.



Third Level Development Tasks

The practical testing of the project happens in the final stage. The application development leads to this. It involves tasks like creating test cases, writing test summaries, coding, drafting bug reports, and actually running tests as shown above. Some dates may overlap due to the constant testing and implementation of the program. By the 22nd of September we would have finished the first and we need to work on building a design model and evaluate and resolve the risks for the second phase. Then we will shift to the third phase. Meaning for the third phase we will need to start working on implementing our detailed design using a Scrum. Based on our experience working with different software system projects the third phase is the most critical. It is also the most time consuming, mostly due to it being hands-on and actually building the software system. Although integration and testing can be exhausting they can be done within a small time frame.

Task	Start	End
Requirements	08/09/2022	22/09/2022
Design	04/10/2022	19/10/2022
Implementation	21/10/2022	18/11/2022
Deployment	21/11/2022	1/12/2022
Testing	21/11/2022	1/12/2022

Requirements

Summary

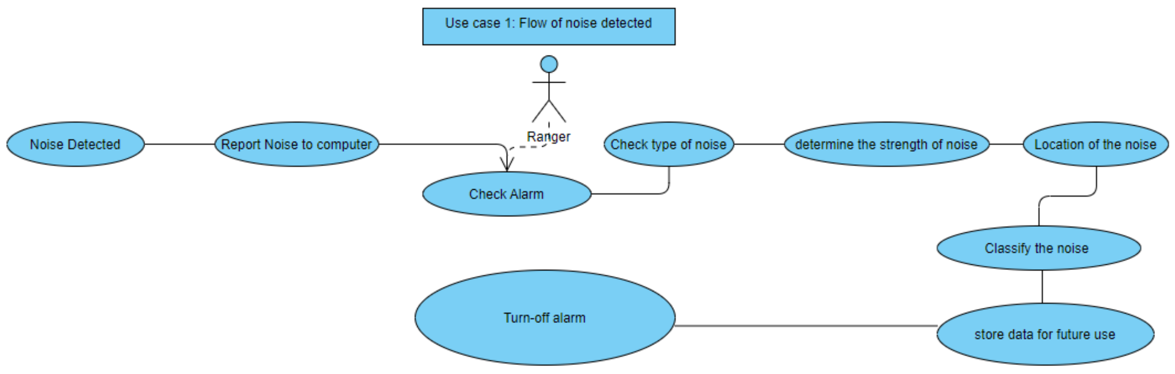
The system is an animal detection system whose main purpose is to send an alert to park rangers if a mountain lion is detected within 5 miles of the detection sensor. The system is designed to distinguish between various types of animal noise and send the noise type to a control computer located in the park ranger station. The control program will display several types of reports upon request. Reports showing the date and classification of the noise whether it be definite, suspected, or false. It will also show the specific location of the detection from the sensor location. This system will have a graphical report showing detections within 2 miles of the park, and will also have an option where it will show the classifications made by the rangers.

Performance-Related Requirements

There will be a detection sensor throughout the park, the sensor will transmit the recorded noise to the control computer to alert the rangers. Park rangers will determine the type of noise, and the system will display the loudness of the noise and its location. This information will be stored for future access. Only the park ranger who controls the computer can turn off the alarm.

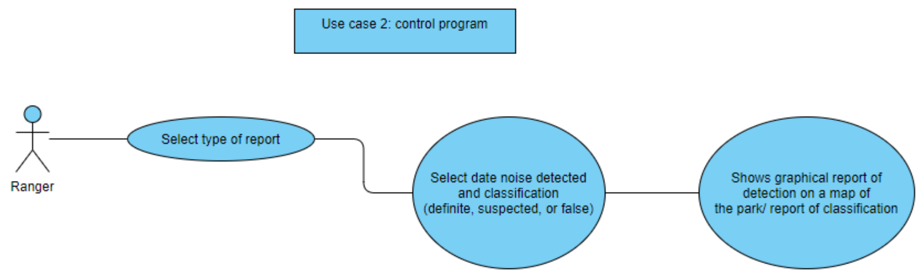
User Cases

Case 1 - Flow of Noise Detected



In the first case, the noise detected by the sensor will be sent to the control computer to trigger an alarm, alerting the ranger (user) to check the recorded noise. The user then classifies the noise as definite, suspected, or false based on the noise detected. This information will be stored in the database as a report for the second use case. Only authorized users (Rangers) are allowed to have access and the ability to turn-off alarms.

Case 2 - Control Program


Use case 2: control program
Ranger
Select type of report
Select date noise detected and classification (definite, suspected, or false)
Shows graphical report of detection on a map of the park/ report of classification

The purpose of the second use case is to display the previous reports of authorized personnel (Park Rangers). Rangers can select the time of the date when there was a detection of noise and the classification accompanying it. This report will show the sensor that detects the noise on the map and there will be a graphical display of this report on the screen.

Discussion of requirements analysis:

Functional Requirements	Allocation of Requirements	Increment/evolution	Requirements to be Implemented for Increment/evolution
Place noise detection sensors within an area of 5 square miles	Hardware	Evolutionary	Will require the use of a sensor that transmits sound to the software intensive system.
Device should be able to detect various types of animal noises	System Software	High Priority- must be implemented first	Will require a working device that detects noise integrated with a program that determines the loudness of the noise to trigger the alert.
Alert messages be sent to the controlling computer based on the type of animal noise detected and the strength of the detected animal noise	System Software	High Priority- must be implemented first	Part of the Software intensive system functionality.
Alert messages will contain the type of noise detected,	System Software	High Priority-	Part of the Software intensive system

the strength of the detected noise, and the location of the detected noise to within 3 meters.		must be implemented first	functionality.
The control program on the controlling computer will sound an alarm whenever An alert message is received from the animal detection system. The alarm will Continue until the ranger turns it off. Once the alarm is turned off, it will not sound again until another, separate noise is detected at a different location.	For us the alarm will be part of the application so it will be a Software Application.	Mid Priority - will be implemented next after highest priority has been proven functional	This is mainly through Web-application which will be designed to set Alarm whenever alert is received.
The controlling computer will save all mountain lion alerts received within the last 30 days. It will save a summary of alert information for data older than 30 days, but received within one year.	Software Application	Mid Priority - will be implemented next after highest priority has been proven functional	This will be part of the Web-application functionality because the Web application is signaling to save the data into the database.
The control program on the controlling computer will allow the ranger to classify each alert as definite, suspected, or false, indicating the probability that a real mountain lion was detected.	Software Application	Mid Priority - will be implemented next after highest priority has been proven functional	This will be part of the Web-application functionality to perform specific tasks such as classification.
The control program will allow the ranger to request several Reports: A report showing al l mountain lion detections by date detected and by classification (definite, suspected, or false) A report showing all mountain	Software Application for performing the tasks System Software for Databases	Low Priority- last to be implemented	Part of the web-application to display landing page, login page and report page. Will be implemented last once all the requirements has been proven

lion detections at a specific sensor location. A graphical report showing detections on a map of the park and areas within 2 miles of the park. A report showing detection classifications by ranger.			functional.
The system should be developed in a way that would allow it to be easily reconfigured for other parks in the State of California.	Software Application, System Software and Hardware.	Evolutionary	This will include all systems because we have to keep up to date with the latest technology, fix security issues that may arise and add more features.

Implementation & Testing

First, we will test the noise detection sensors, this will be achieved by integrating our programs into raspberry pi hardware. Once, that's been established as operating correctly. We will then move on to the Alert messages testing to see if an alert is being sent after the sensors have detected noise. We would then move on to making sure that controlling computers are receiving these alerts, it is important because this is how the rangers will be able to classify the noise detected and log the information to be sent to the database. Lastly, we will need to test if the information are being stored in the database and the graphical display is showing the right information when requested by the Ranger.

Work Done on Requirements

Meeting #	Date & Time	Requirements	Notes
Initial	05/09/2022 @ 11:00 A.M. - 12 P.M.	Hardware	Raspberry Pi, Other Hardwares, etc.
2nd Meeting	08/09/2022 @ 6:15-6:35 P.M.	System Software	System requirements i.e. database

3rd Meeting	16/09/2022 @ 11:00 A.M. - 12 P.M	Software Application	Discussed software application and requirements.

Initial

In our first meeting as a group, we discussed the hardware requirements for the project. We decided to use a Raspberry Pi to implement data transfer between the detection system and the database. We also agreed that our group will use this as a sensor device that detects noise to assist in the implementation of the detection system. Lastly, the computer that will display the output of the system.
Number of initial Requirements discussed: 3

2nd Meeting

Now that the hardware requirements have been established. In our second meeting, we discussed the overall design of the product. Our group decided that we will be needing a database for our log system to store the previous detection's information. To do this, we all agree that we need to allocate most of our efforts to the detection system; our alert program.
Number of requirements discussed: 2

3rd Meeting

Finally, because this program needs a screen. We discussed the need for a UI to display the landing page, login page, and report page. Initially, we wanted to have a registration system within the log-in page, as only authorized personnel would have access to alerts and reports. Ultimately, we all agreed that there was no need for a registration system, as we only needed to implement an admin page for supervisors to add other workers to the system.
Number of requirements discussed: 4

Final number of total requirements

Overall, there are 9 total requirements discussed in our analysis, which are required for our decision to move on to the top-level design for now. A detailed summation of these requirements can be seen under the "Discussion of requirement analysis" section of this write-up.

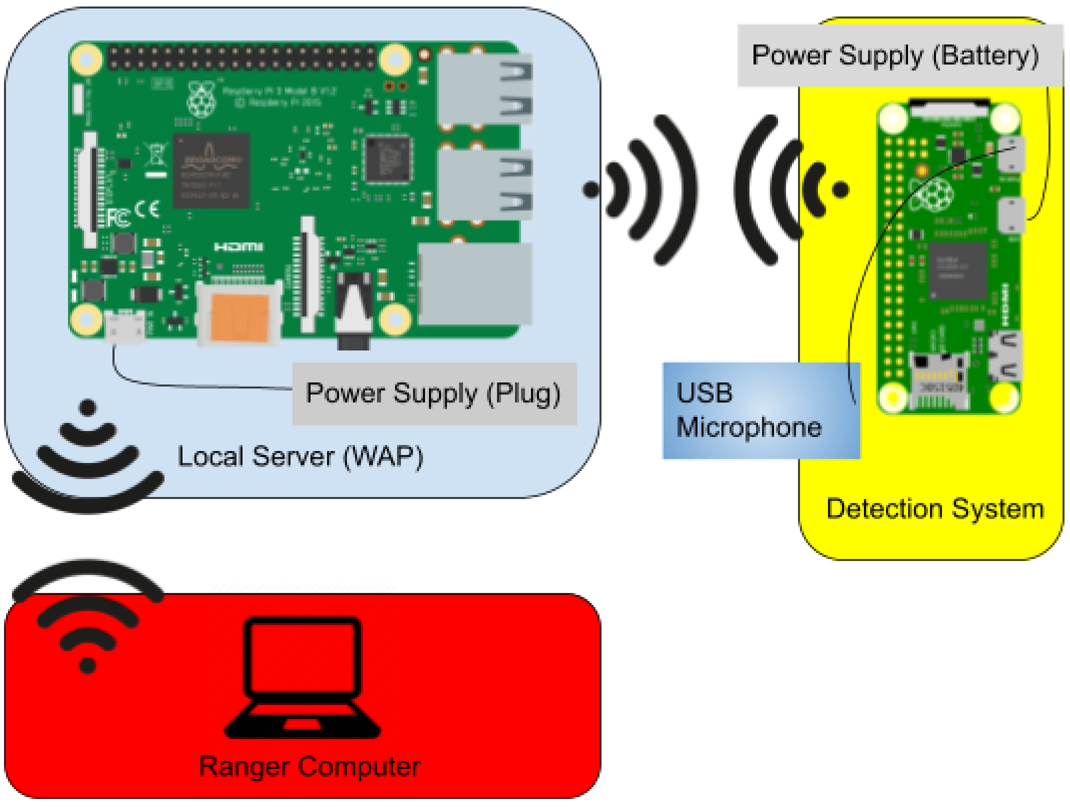
Design

Top Level Architecture

Hardware Design

Electronics

The electronics design for this project heavily relies on the Raspberry Pi platform. We will use two different Pi devices for the initial prototype, one for detection and one to act as a server within the rangers cabin. This design has the possibility of being scaled to up to 50 different detection units, and past that at the loss of some performance (i.e. connection drop issues).

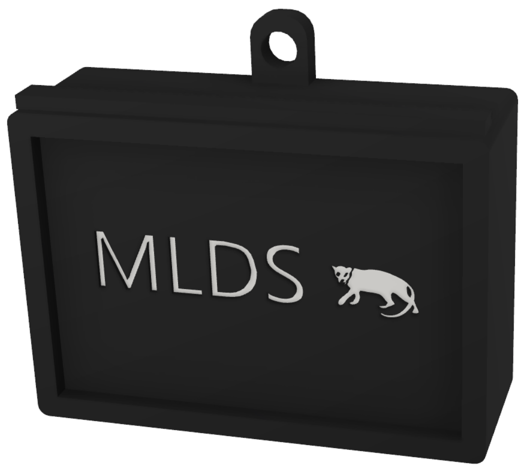


High-Level Electronics System Design

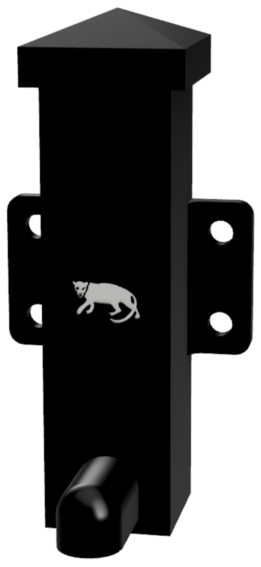
Since this is only a prototype with limited funding, we have omitted the necessary receivers/transmitter modules for long-range WIFI connections. With the addition of such long-range wifi modules, the system should be able to function well in the often challenging and vast park environment. It is also required for the detection device to be able to sense and record the audio of its surroundings. We will be utilizing a simple USB microphone to do this sensing, it provides us with a continuous audio signal for analyzing. For the Pi to function it requires a power supply, of which can often be sparse in parks, so we will be using a battery pack. In future updates it could be useful to attach a solar panel to remove the need for recharging.

Chaises

To increase the life of the electronics and also so they are easily identifiable, we will 3D print housings to encapsulate them. We will generate these housing whilst keeping the environment they are to be placed in in mind. For them to operate properly they will have to have their microphones exposed, but we also need to make sure they don't become damaged. Following are the designs for both the server and detector chaises.



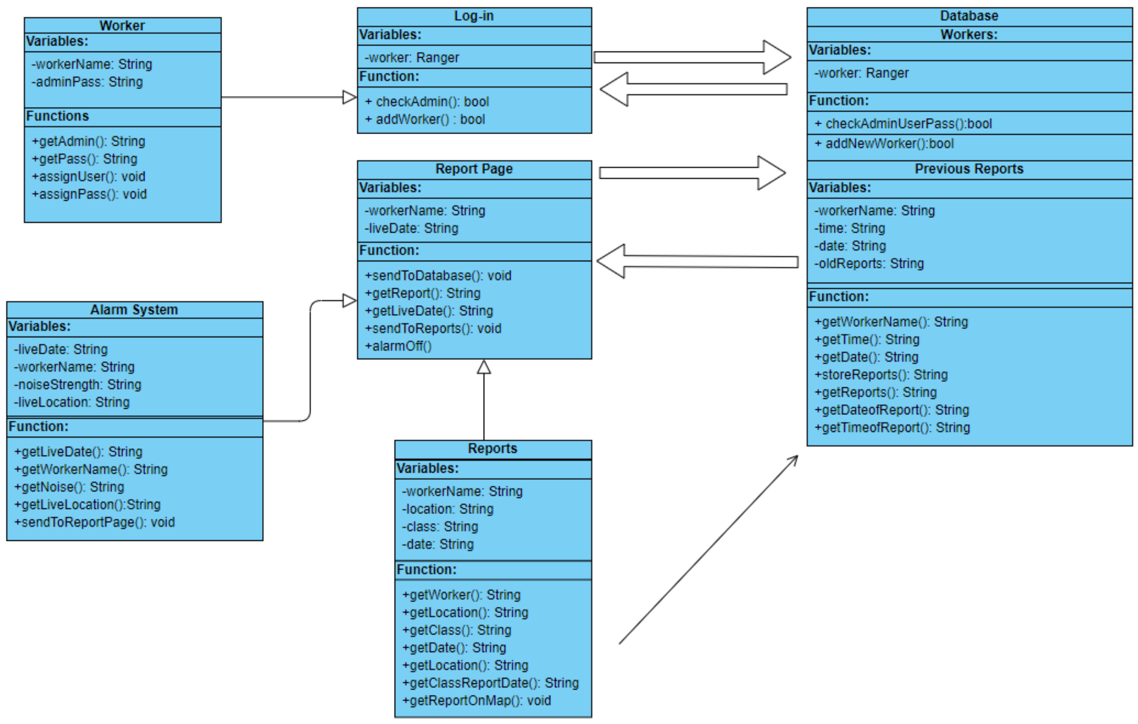
Server Case



Detector Case

UML Diagram

Initial



Worker Class

Variables:

- workerName = name of the ranger or admin
- adminPass = this holds the password for the admin

Function:

- getAdmin() = will fetch the admin username if it exist in the database
- getPass() = will fetch admin password if it exist in the database
- assignUser() = this gives the admin the power to assign a worker class username and send it to database
- assignPass() = this gives the admin the power to assign a worker class password and send it to database

Log-in Class

Variables:

- worker: worker class to create worker object (ranger)

Function:

- checkAdmin(): boolean functioning. If true, that means that admit info exists in the database and will grant access
- addWorker(): Admin will add a new worker and will return a bool if the registration of the new worker has been successful

Reports Class

Variables:

- workerName: name of the Ranger that submitted the report
- location = the location where the animal was detected
- class = classification of the report (Definite, False, Suspected)
- date = the date of the report

Function:

- getWorker() = retrieve the worker that submitted the report
- getLocation() = retrieve the sensor which detected the animal
- getClass = prints the classification to the screen
- getDate() = will show the dates of all submitted reports
- getClassReportDate() = prints the date of the reported classification on the screen
- getReportOnMap() = will show the location on the map display

Alarm System Class

Variables:

- liveDate: this will show the actual date when the animal was detected
- workerName: will alert whoever is on shift that an animal was detected
- noiseStrength: will get the measurement of the noise from the sensor
- liveLocation: will show which sensor detected the animal

Function:

- getLiveDate() = will save the to date of when the animal was detected
- getWorkerName() = will send alarm to the worker’s computer
- getNoise() = will retrieve the noise from sensor.
- getLiveLocation() = will save where the animal was located and which sensor detected it
- sendToReportPage() = will send the information to report page.

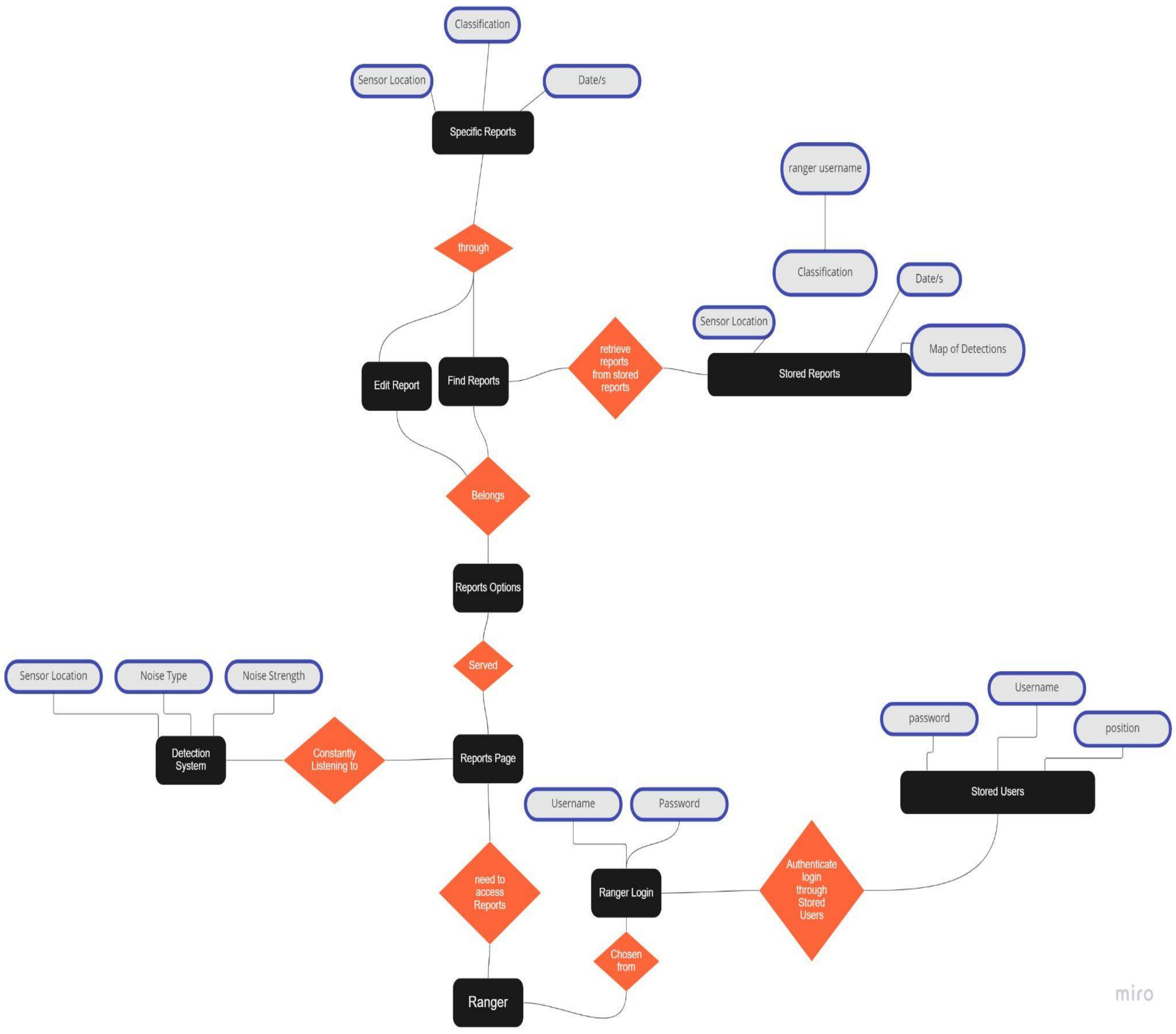
Final

As we developed the project we realized that an OOP style wouldn’t best suit the project. We instead developed a layered system that utilized an API layer with simplistic functions to deliver the data, to a more complex React GUI that has the ability to transform data.

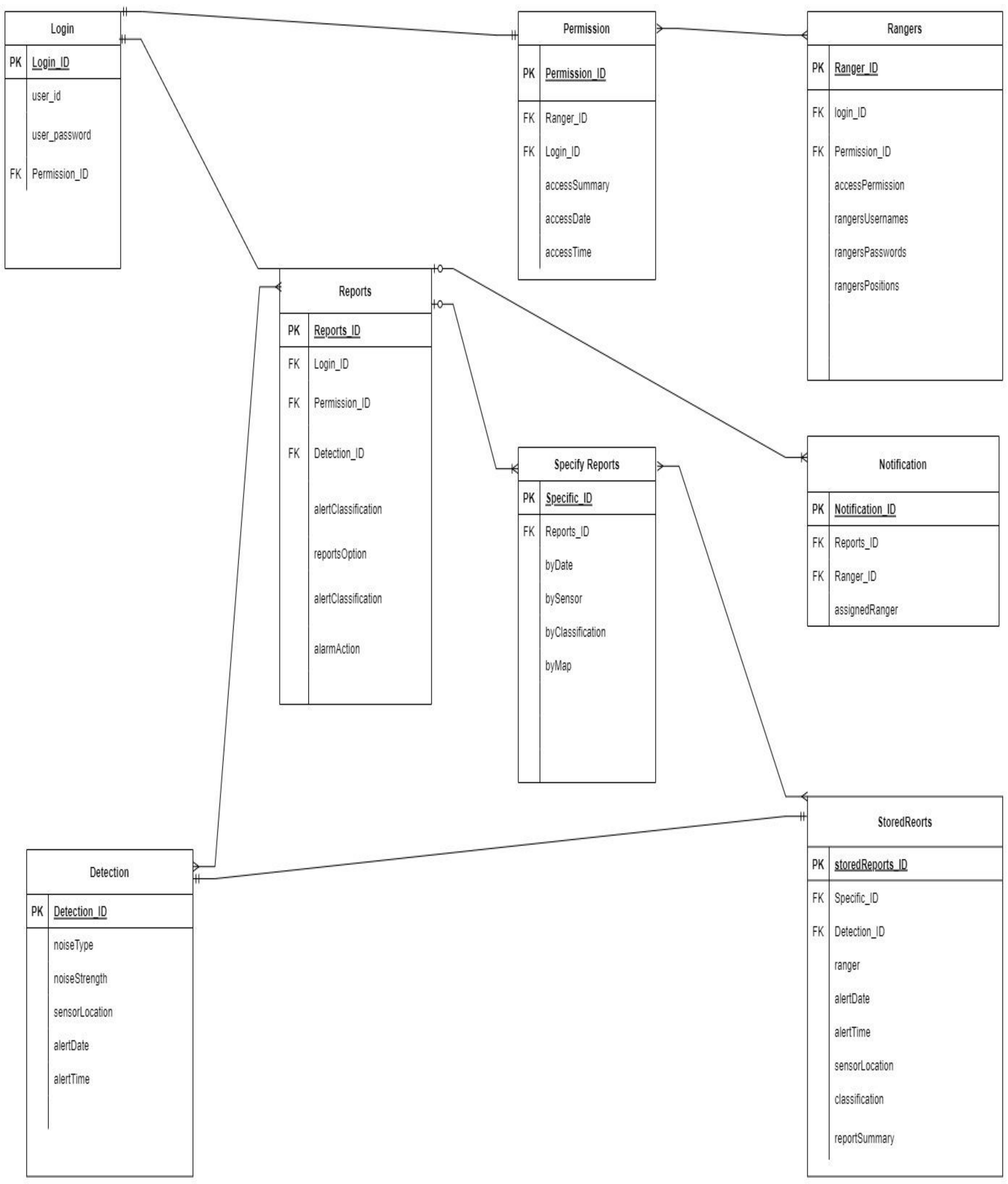
Database Design

Initial

One of the most important components of our system is the Database. Not only will we be receiving data from the database through the main Computer; we will also design our system to store mountain lion detections directly from our raspberry pi to the database locally without the need of a main computer. The main Computer will be the head of operations in our Mountain Lion Detection System and it will perform everything from classification, retrieving data/displaying data and adding new users to the database. Below is an initial entity relationship diagram which showcases the way the database will interact with the rest of the system. Keep in mind that rectangles are entities, ovals are attributes and diamonds are relationships. For better quality this is the link to the diagram
Functional Decomposition, Online Whiteboard for Visual Collaboration (miro.com)



Below is the Entity Relationship Diagram with its tables, attributes, and relationships between tables. While it’s based on the diagram above we had to make few changes to ensure its functionality, relativity to what programming a system like this would look like and just how the tables/classes interact with each other.



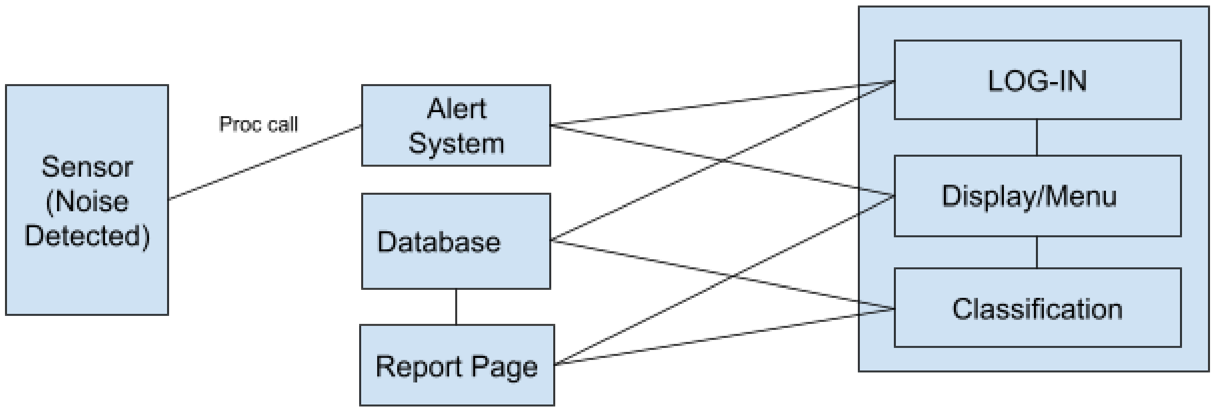
.

Final

We started with a more encompassing relational DB design. As we implemented we managed to reduce it further helping better utilize the limited hardware, following the core engineering concept KISS (Keep it simple stupid!). The system now simply has two different collections within mongoDB; one for Login details and another for Detections.

Architecture

Initial - Event Based

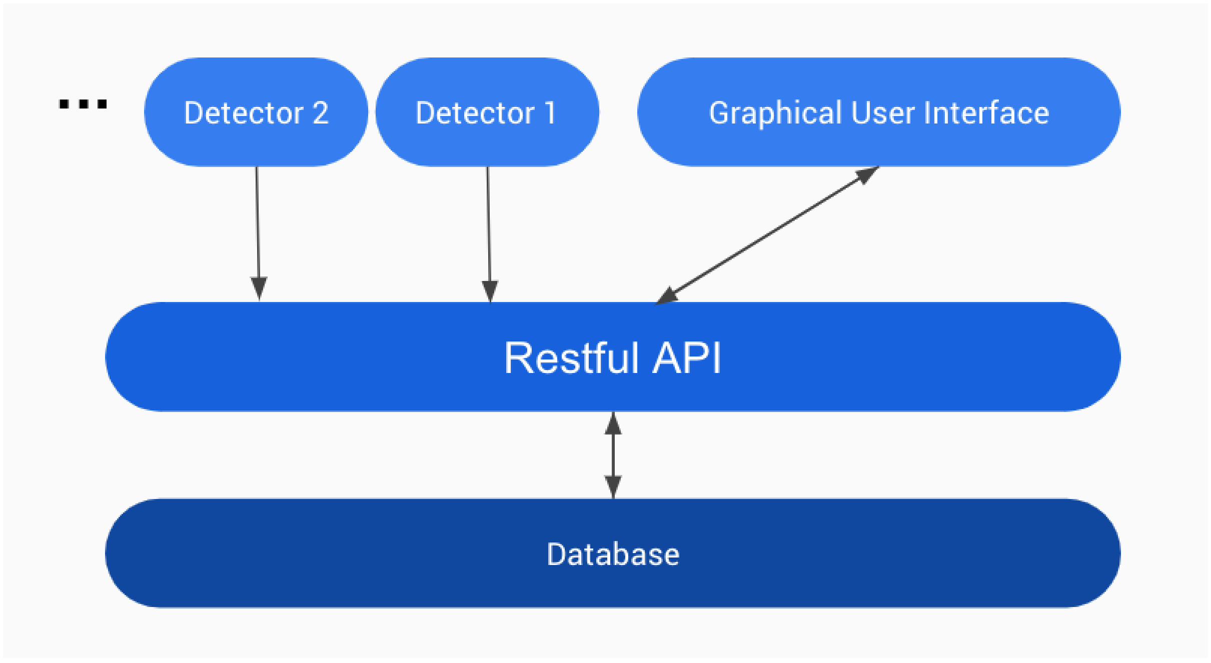


Module	Users	Description
Sensor	Admin	Activates the proc call to turn on the alarm system
Log-in	Admin Worker	Admin: main role is to register worker in system Worker: Log-in to access Display/Menu
Display/Menu	Worker	Worker: May choose to go to classification page or view previous report
Classification	Worker	Worker: Page where worker where worker classify detected noise
Report Page	Worker	Worker: Page that show graphical reports of map and previous reports

Database	Admin	Admin: Store/fetch data information from/for log in, classification, and report page

Final - Layered

As we started implementing the project we realized that the system could more easily be created following a Layered architecture. The current implementation has 4 main components; Detectors, the GUI (Graphical User Interface), the Restful API and the Database.

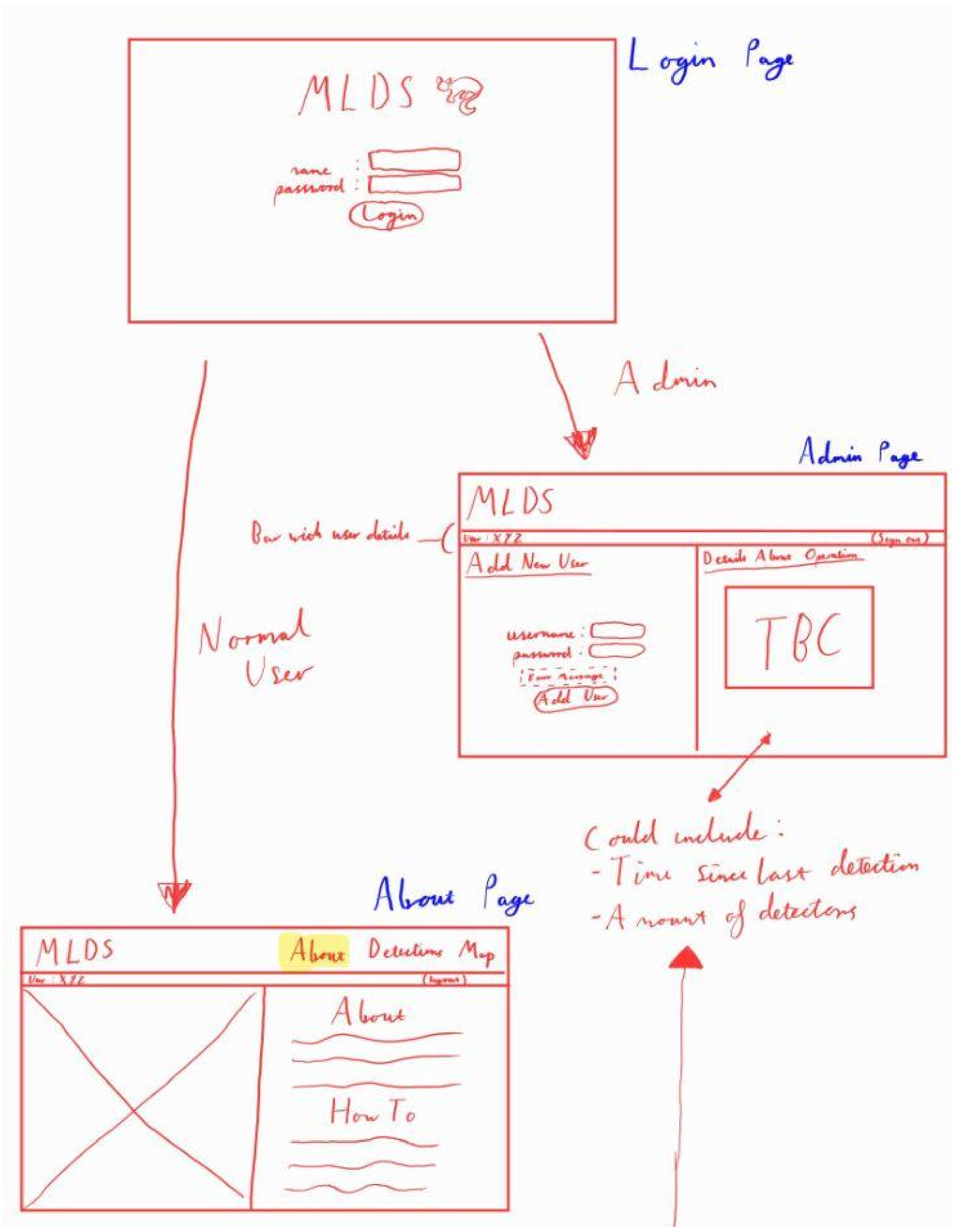


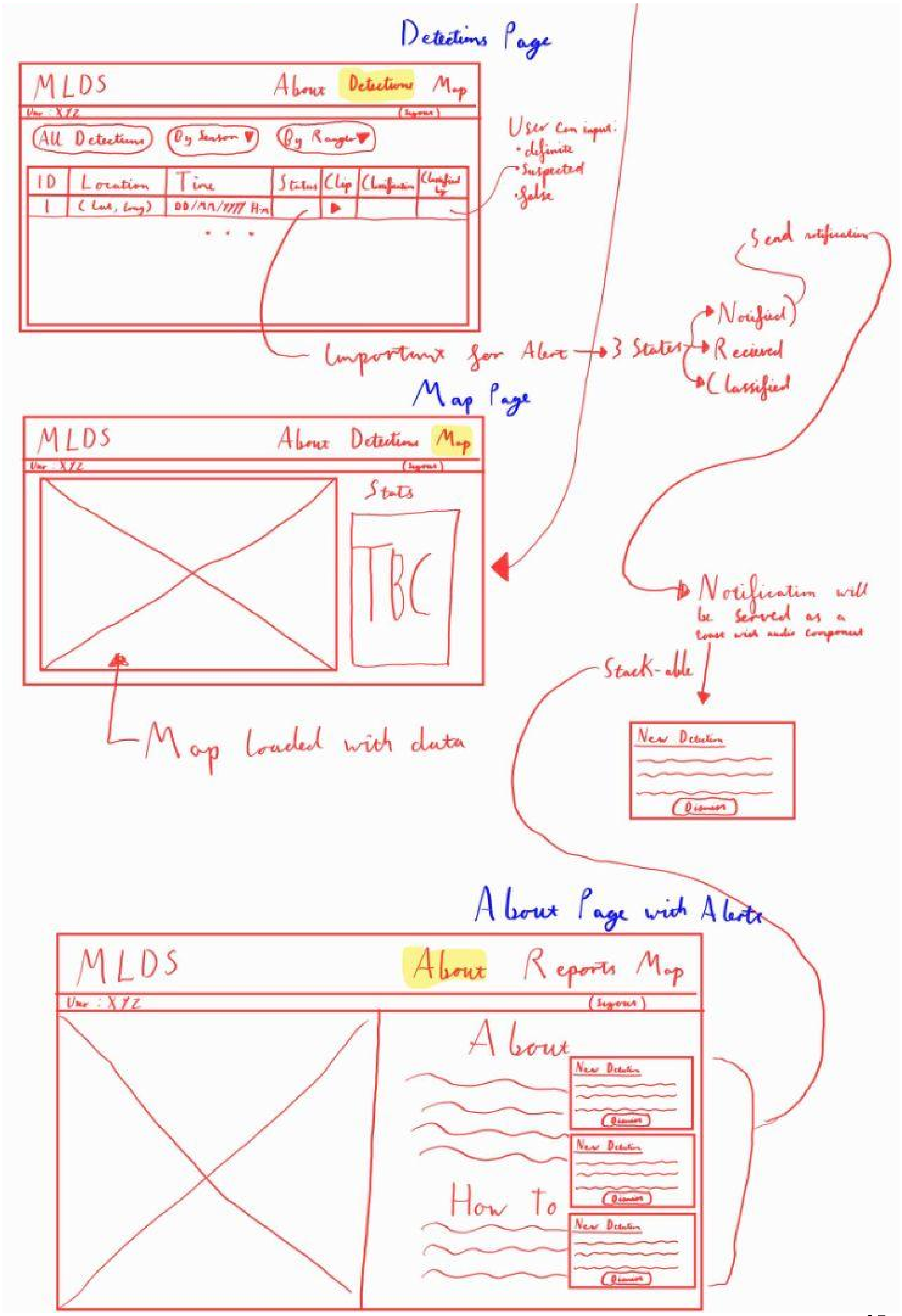
Component	Description
Detector (1,2…)	Devices to be placed in the park that send messages to the API when sound levels reach a threshold .
Restful API	The brain of the system, responsible for managing requests and saves from both the detectors and users.
GUI	The component which allows a Ranger to interact and analyze the data delivered by the API. Also housing admin support and functionality.
Database	The database is what the system uses to store and organize the required system data. Both Login and detection details.

UI Prototype

We designed a multi-page UI for the rangers to interact with the system. We kept the prior disclosed requirements in mind whilst creating the following 5 page design.

UI Traversal Wireframe



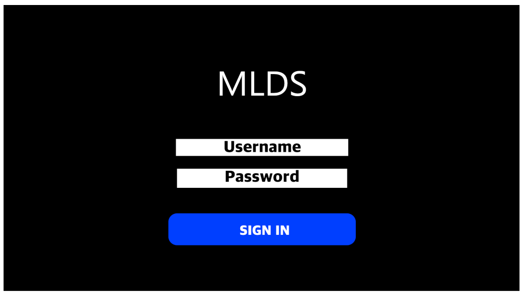


Detections Page
MLDS
About
Detections
Map
User : XYZ
Logout
All Detections
By Season ▼
By Ranger ▼
ID
Location
Time
Status
Clip
Classification
Classified by
1
(Lat, Long)
DD/MM/YYYY H:m
User can input:
• definite
• Suspected
• false
Important for Alert
3 States
Notified
Recieved
Classified
Send notification
Map Page
Stats
TBC
Map loaded with data
Notification will be served as a toast with audio component
Stack-able
New Detection
Dismiss
About Page with Alerts
Reports
About
How To

Page Breakdown

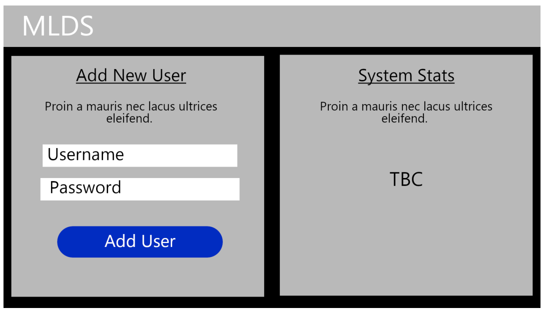
Login Page

The system requires us to keep track of which ranger makes which classifications. A simple way of doing this is implementing a user system. To track the current user it's necessary to get them to login with their credentials at the start of use. This can be achieved by using a simple landing page, including; username input, password input and submission button.



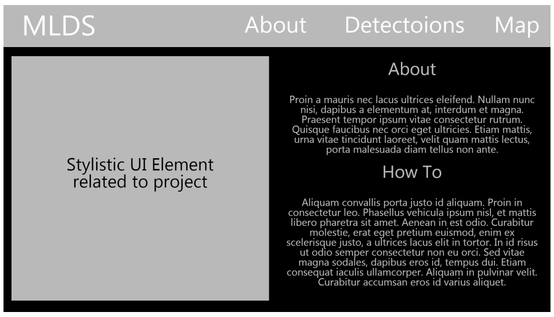
Admin Page

The system should be able to handle the addition of new rangers. This core feature should be controlled by an administrative figure. We will therefore restrict the screen to a user with the specific login details. The page will have a simple form for inputting new user details, it will also display stats to allow the admin to gauge system performance.


MLDS
Add New User
Proin a mauris nec lacus ultrices eleifend.
Username
Password
Add User
System Stats
Proin a mauris nec lacus ultrices eleifend.
TBC

About Page

After a ranger logs in using their given credentials, they will be greeted by an about page. This page will hold information about how to properly use the application and also a graphical element to add interest to the UI.

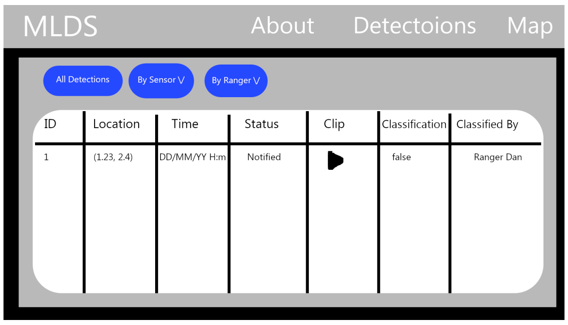


Proin a mauris nec lacus ultrices eleifend. Nullam nunc nisi, dapibus a elementum at, interdum et magna. Praesent tempor ipsum vitae consectetur rutrum. Quisque faucibus nec orci eget ultricies. Etiam mattis, urna vitae tincidunt laoreet, velit quam mattis lectus, porta malesuada diam tellus non ante.

Aliquam convallis porta justo id aliquam. Proin in consectetur leo. Phasellus vehicula ipsum nisl, et mattis libero pharetra sit amet. Aenean in est odio. Curabitur molestie, erat eget pretium euismod, enim ex scelerisque justo, a ultrices lacus elit in tortor. In id risus ut odio semper consectetur non eu orci. Sed vitae magna sodales, dapibus eros id, tempus dui. Etiam consequat iaculis ullamcorper. Aliquam in pulvinar velit. Curabitur accumsan eros id varius aliquet.

Detections Page

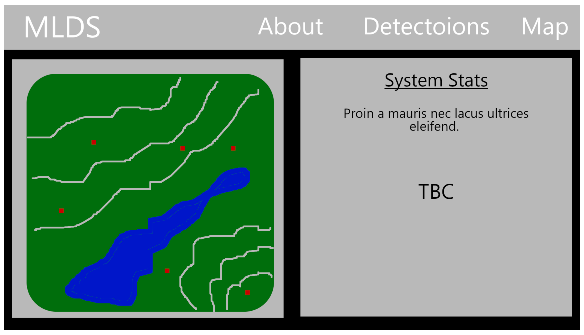
This page will be accessible for a basic user by selecting it in the navbar, after logging in. The page will have filter options to appease the requirements (i.e. by ranger). These filters will edit the detections shown in the table. The rangers will also be able to make classifications on this page.



ID	Location	Time	Status	Clip	Classification	Classified By
1	(1.23, 2.4)	DD/MM/YY H:m	Notified	▶	false	Ranger Dan

Map Page

This page will use the different detection's location data to plot the incidents on a map of the park. This page will also include details about the amount of cat detections etc. so that the ranger can have some quick statistics on the current system levels.



System Stats

Proin a mauris nec lacus ultrices eleifend.

TBC

Work Done on Design Summary

Meeting #	Date & Time	Focus	Notes
4th Meeting	30/09/2022 @ 11:00 A.M. - 12 P.M.	Hardware	Decided on hardware design
5th Meeting	08/10/2022 @ 6:00-6:30 P.M.	OOP	Discussed GUI and how it will interact with backend/DB
6th Meeting	14/10/2022 @ 11:00 A.M. - 12 P.M	DB and Misc	Discussed DB and also reworked hardware.

Code and Tests

Unit Test Plans

Unit Testing is a White box testing technique and white box testing is a methodology where the test cases are mostly derived from examining the code and the detailed design. Below are all the components that we need to ensure on testing as we go along to make it easy to locate errors when they do arise.

Unit Test ID	**Component**	**Input**	**Expected Output**	**Details**
MLDS#1	Database access	Username Password	Access/ Deny	Test the access rights on MongoDB to ensure only registered people have access to the Reports page.
MLDS#2	Database access	Username Password	Stored	Test the ability to add new members to the Rangers DB by storing their ID's
MLDS#3	Database storing object	classification	Stored	Test the store/add function of the DB to ensure it store ranger's classification
MLDS#4	Database display object	Report Date/s, location, classification , map.	The Report/s	Test the search/scan and print functions to ensure its ability to find and print needed reports
MLDS#5	Raspberry Pi noise recording object	From	Current noise	Test the Raspberry Pi program range function to ensure it is constantly listening and only recording noise if it passes the threshold.
MLDS#6	Raspberry Pi noise recognition object	Recorded Noise from Microphone	Noise Classification	Test the range function in the Raspberry Pi program to ensure it classifies the received recording.

MLDS#7	Raspberry Pi sensor Location accuracy	From Raspberry pi assigned ID and it’s designated location	Designated location	Check the Raspberry Pi program string class function to ensure it saved the right data.
MLDS#8	Raspberry Pi Date and Time	From datetime class and module	Date and Time	Test if the datetime to string method is working and the program is converting the current date time object to a different string format.

Unit Test Review Notes

Up to 10/20/2022 we have been able to code and unit test those components.

Unit Test ID:	**Date tested:**	**Notes:**
MLDS#1	09/24/2022	Using MongoDB we have created a username and password for each active ranger. The only difficulty we faced is figuring out how to organize the rangers into their own database to realistically keep track of essential data.
MLDS#2	09/24/2022	Using MongoDB we added rangers and assigned usernames and passwords so they can access the reports page.
MLDS#3	9/27/2022	Using MongoDB the Ranger was able to Store/add their classification/summary into a sample database.
MLDS#5 partially	10/04/2022	We tested the microphone and ran a quick recording test to see if it is compatible with our Raspberry Pi. We haven’t decided on the threshold yet.

MLDS#6 partially	10/09/2022	We were looking at voices to add to our library which Mountain Lions typically make. It is extremely challenging to find a pattern yet.
MLDS#7	10/09/2022	Gave the raspberry pi a designated location on a virtual setting/map to display the origin of the noise.
MLDS#8	10/12/2022	Tested that the accurate date and time is being recorded in the program.

Unit Test Results

From the components we tested above these were the results:

Unit Test ID	**Passed**	**Not Passed**	**Retested**	**Comments**
MLDS#1	✓		✓	No need to go fix anything this is already done.
MLDS#2	✓		✓	No need to go fix anything this is already done.
MLDS#3		✓	✓	It works but we only ran it through a sample database. Test again once finished with MLDS#B
MLDS#5		✓	✓	We haven't yet decided on the threshold but we have everything ready. Retest after establishing the threshold and entering it in the function.
MLDS#6		✓	✓	This has been the most difficult component to get right so far due to researching and the machine learning to recognize the sounds actual Mountain Lions make. Retest as you expand your libraries. Might need to change the code a little too.
MLDS#7	✓		✓	No need to go fix anything this is already done.
MLDS#8	✓		✓	No need to go fix anything this is already done.

Integration Test Plans

Technique from the Black Box testing methodology, Involves testing the integrated functionality of the complete system. The difference between Unit testing and Integration testing is that Unit testing is done at Component level/Unit level while Integration testing is done when the components are being combined. We are using IBM’s Rational Integration tester and FitNesse’s open source project.

Integration Test ID	**Function**	**Test Tools**	**Details**
MLDS#9	Database-Raspberry Pi	Stubs	Test the connection between Raspberry Pi and the DB.
MLDS#A	HTML-DB-UI	Driver/ Stubs	Testing to see that the entered username and password from HTML matched the one in the database, then verifying it to be transferred to the Reports Page.
MLDS#B	User Interface-Database	Driver/ Stubs	Testing to check that we are getting the right output based on the option chosen from the Reports page specification.
MLDS#C	Raspberry Pi to Database packets	Driver/ Stubs	Test to see if the installed MongoDB on our Raspberry Pi is able to transfer and save the alert data directly into the database.
MLDS#D	Raspberry Pi packets to main Computer	Stubs	Test the HTTP send and receive server to ensure that it is delivering the alert message to the main Computer.
MLDS#E	HTTP-Main Computer Alarm	Driver	Test if the alarm turns on whenever data is received through the HTTP Server.

Integration Test Review Notes

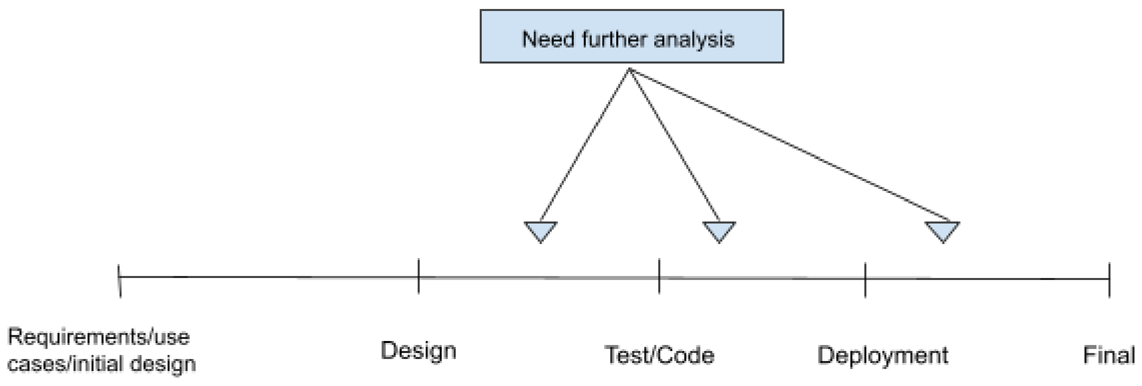
Integration Test ID:	Date tested:	Notes:
MLDS#9	09/28/2022	We installed MongoDB on Raspberry Pi and created a local connection as well as a test to see if it takes data from Raspberry Pi and stores them into the Database. However we didn’t use the final version of the data from the processed recordings.
MLDS#A Partially	10/05/2022	Using a Driver testing tool due to missing low level components from each module. Still needs a lot of work and better design.
MLDS#B	10/9/2022	Using a Driver testing tool because due to missing low level components from each module.
MLDS#C Partially	09/28/2022	We installed MongoDB on Raspberry Pi and created a local connection as well as a test to see if it takes data from Raspberry Pi and stores them into the Database. However we didn’t use the final version of the data from the processed recordings.
MLDS#D	10/17/2022	We sent a sample data message from Raspberry Pi to the Main Computer using HTTP.
MLDS#E	10/19/2022	We tried to figure out how to turn on and control an alarm after receiving the alert message but we still need to do a lot of work.

Integration Test Results

The results in here are due to the used tools which ignore certain missing components such as Stubs/Drivers in order to check the concepts and provide feedback to the Developers.

Integration Test ID	**Passed**	**Not Passed**	**Retested**	**Comments**
MLDS#9	✓		✓	Works, but we still need to get full and accurate data on the database.
MLDS#A	✓		✓	Works, but we still need to get full and accurate data on the database.
MLDS#B	✓		✓	Works, but we still need to get full and accurate data on the database and add the map option to the UI and connect it to the database.
MLDS#C	✓		✓	Works, but we transferred a sample of an alert message data to be stored in the database due to the holdback on MLDS#B.
MLDS#D	✓		✓	Works, but we transferred a sample of an alert message due to the holdback on MLDS#B.
MLDS#E		✓	✓	We still haven't got a hold of how the alarm should operate or how to turn it on once an alert message is received.

Risk Status/Areas Needing Further Analysis



Design:

- May change depending on the result of test
- End product followed correct standards in implementation. API can only be access using GET Requests

Test/Code:

- Test and Code may change depending how much changes are made in the design
- Tested UI and Detector with different technically inclined people.

Deployment:

- Deliverable and deployment may take longer if changes are implemented.
- Deployment had to be delayed due to incomplete components. i.e . Integrations of all the 3 subsystem: Database, UI, Detection.

Independent System Test

Test Plan

To test the project not just for our team but for the team we will be testing their project as well; we decided to go by the requirements given to each team based on their chosen project. For each of the requirements we decided to follow these three rules:

1) Check if the requirements are achievable and within the bounds of what's achievable by small groups of students.
2) That each requirement was met despite the difference in execution
3) If the requirement was not met rate the team based on how far the progress to meet those requirements went

   For number one specifically some of the requirements for our part were beyond the realm of achievability; there was one requirement regarding putting sensors in a 5 miles long area. Not only would each sensor cost money that we don't have, we have to obtain permission to plant those sensors and would need more people to sustain such a system.

   When it came to test tools we came to the conclusion that some form of testing was necessary to ensure that the program was running and up to current standards in software development. Whether using an automated API to test the program or Unit testing each part; it is crucial that some form of testing is needed to ensure the functionality of the programs and the system as a whole. Some Informal notes related to test planning activities are: Track the team progress and their string of thought to brainstorm and understand why they chose to pursue certain methods. As well as keeping an open mind and being open to constructive criticism. According to the class notes a requirement is testable if an objective and a feasible test can be conducted to determine if the requirement is met by the software. The notes also highlight the importance of planning early as soon as the requirements have been identified and the formal test plan should identify test tools, test drivers/stubs, special conditions, input data, expected output data, test procedures, analysis procedures, analysis procedures and pass/fail criteria. Therefore, when following the review notes and comparing to our requirements we came to this we can easily derive the input and know what the expected output should be, the requirements are also well defined and special cases are well outlined. We decided to create a checklist to keep track of fulfilling the requirements the best we could. Under part b is the checklist we made for our project and it covers what the class notes expects in regards to meeting the requirements and is a summary of the conducted test plan for each requirement.

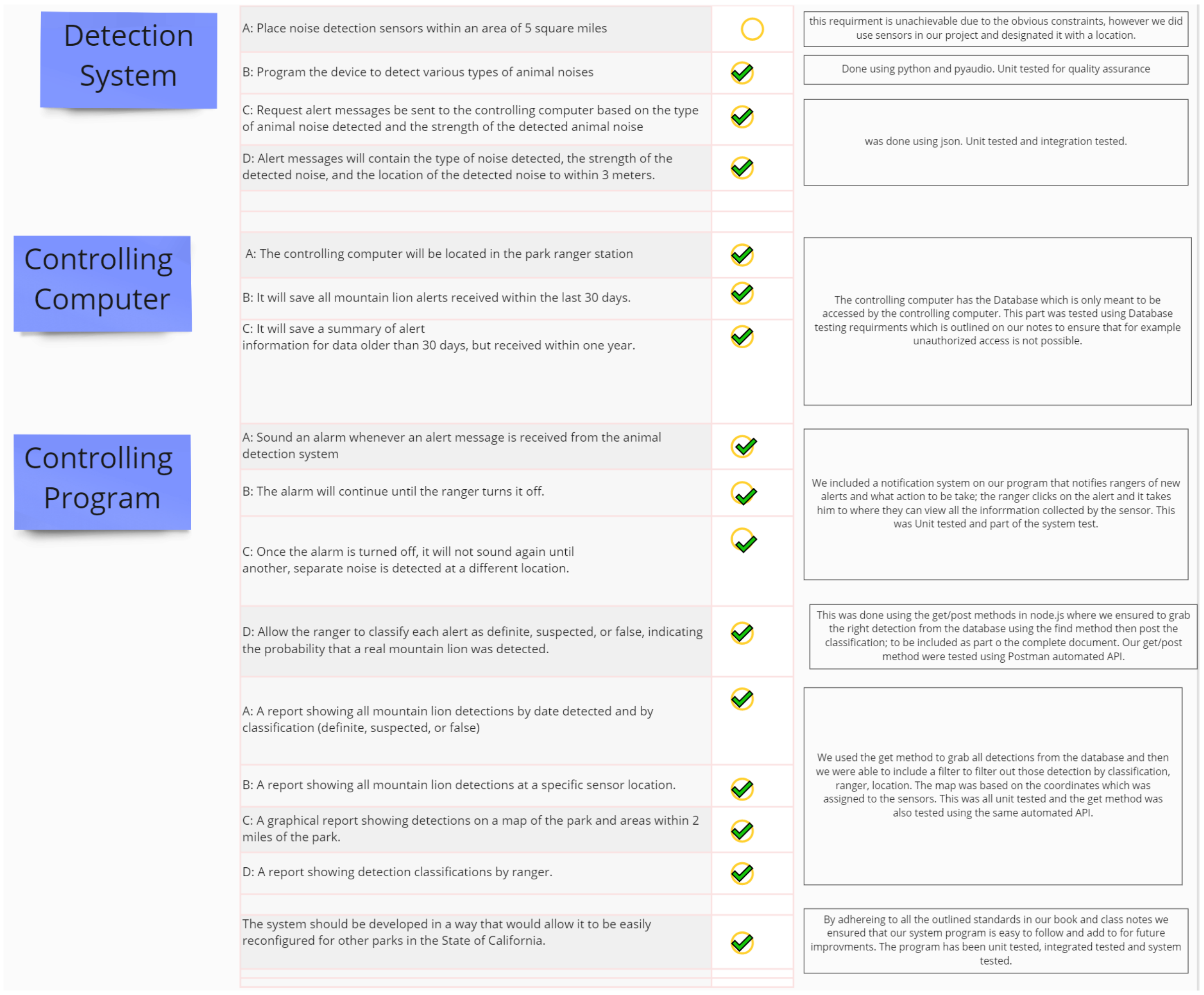
Test Procedures

For our project we stuck to the requirements and operated our testing based on it. The requirements themselves are pretty self explanatory. Since the test procedures contain detailed step by step instructions for the set up execution and of each test; we simply can derive those test procedures from the identified requirements and ensure that malfunctions that the requirements doesn't specify are also monitored such as security and other bugs that can be a huge issue in the deployment phase. For each test case we ensured that the opposite of what is defined in requirements doesn't happen. So for example The detection system program should be able to send alert messages when the detection sensors detect suspicious noises. If the program doesn't send these alert messages using json then the test case has failed. When putting the check mark we ensured that not only for example that the ranger can classify the detection but that when classified it will be saved to the document and if it's not saved to the document or if there are any inaccuracies then we failed the test. As shown below; originally we have done all of what's below on paper and I felt the need to instead of taking an intangible picture due to lack of organization that we should just turn it into a table of all our notes.

Description of Test Procedure	Case1	Case2	Input	Expected Results	Prerequisite
Detection System:	1)The device can detect various types of animal noises. 2)The device can send alert messages to the controlling computer based on the type of animal noise detected and the strength of the detected animal noise 3)The device sends alert messages that will contain the type of noise detected, the strength of the detected noise,	1)The device can't detect various types of animal noises. 2) The device can't or is inconsistent when sending alert messages to the controlling computer based on the type of animal noise detected and the strength of the detected animal noise 3) The device can't or is inconsistent or won't send all the outlined information when sending the alert	Different noises	The noise is sent because it's suspicious such as being too loud or sounding similar to a mountain lion, alert messages sent which contain the type of noise detected, the strength of the detected noise, and the location of the detected noise.	For 1 no prerequisite . For 2 and 3 Must detect the noise and meet the suspicion requirements to be sent and recorded

	and the location of the detected noise to within 3 meters.	messages			
Controlling Computer:	1)One controlling computer 2) It will save all mountain lion alerts received within the last 30 days. 3)It will save a summary of alert information for data older than 30 days, but received within one year.	1)Unauthorized access 2)The controlling computer won't save all mountain lion alerts received within the last 30 days. 3)it will not save a summary of alert information for data older than 30 days, but received within one year.	Alerts from Detectio n System	Alerts saved into the database in the controlling computer which are accurate and within one year.	For 1 no prerequisite For 2 and 3 is a connection between the detection system and the database as well as an alert which is based on the detection system ability to detect noises.
Controlling Program:	1)An alarm whenever an alert message is received from the animal detection system 2)The alarm will continue until the ranger turns it off. 3)Once the alarm is turned off, it will not sound again until another, separate noise is detected at a different location. 4)The ranger can classify each alert as definite,	1)No alarm or inconsistency whenever an alert message is received from the animal detection system 2)The alarm won't need a ranger response 3)The alarm won't stop after a ranger's response 4)The ranger can't classify each alert as definite, suspected, or false.Or the classification is not saved or any other inconsistency	1)alert messag e 2)ranger respons e to alert messag e 3)ranger classific ation 4)Filter input by ranger	1)alarm which can be turned off by a ranger and it corresponds to new alert messages. 2)classificatio n by ranger is available and saved to the right document. 3)ability to request all documents within one year and filter through them.	For 1,2 & 3 an alert message which means the prerequisite is the detection system similar to the controlling program. For 4 it will be the database/ controlling computer which should have

	suspected, or false. 5)Alerts can be filtered based on detection classifications, location and ranger 6)A graphical report of based on the location of the detection	5)Alerts can’t be filtered based on detection classifications, location and ranger 6)No graphical report/map shows the detections based on the location on the map.		4)a graphical report of the detections on a map.	saved these alerts to the database. For 5 and 6 the information from the alerts which are stored in the database and #4.



Detection System	A: Place noise detection sensors within an area of 5 square miles	○	this requirment is unachievable due to the obvious constraints, however we did use sensors in our project and designated it with a location.
	B: Program the device to detect various types of animal noises	✓	Done using python and pyaudio. Unit tested for quality assurance
	C: Request alert messages be sent to the controlling computer based on the type of animal noise detected and the strength of the detected animal noise	✓	was done using json. Unit tested and integration tested.
	D: Alert messages will contain the type of noise detected, the strength of the detected noise, and the location of the detected noise to within 3 meters.	✓	
Controlling Computer	A: The controlling computer will be located in the park ranger station	✓	The controlling computer has the Database which is only meant to be accessed by the controlling computer. This part was tested using Database testing requirments which is outlined on our notes to ensure that for example unauthorized access is not possible.
	B: It will save all mountain lion alerts received within the last 30 days.	✓	
	C: It will save a summary of alert information for data older than 30 days, but received within one year.	✓	
Controlling Program	A: Sound an alarm whenever an alert message is received from the animal detection system	✓	We included a notification system on our program that notifies rangers of new alerts and what action to be take; the ranger clicks on the alert and it takes him to where they can view all the inforrmation collected by the sensor. This was Unit tested and part of the system test.
	B: The alarm will continue until the ranger turns it off.	✓	
	C: Once the alarm is turned off, it will not sound again until another, separate noise is detected at a different location.	✓	
	D: Allow the ranger to classify each alert as definite, suspected, or false, indicating the probability that a real mountain lion was detected.	✓	This was done using the get/post methods in node.js where we ensured to grab the right detection from the database using the find method then post the classification; to be included as part o the complete document. Our get/post method were tested using Postman automated API.
	A: A report showing all mountain lion detections by date detected and by classification (definite, suspected, or false)	✓	We used the get method to grab all detections from the database and then we were able to include a filter to filter out those detection by classification, ranger, location. The map was based on the coordinates which was assigned to the sensors. This was all unit tested and the get method was also tested using the same automated API.
	B: A report showing all mountain lion detections at a specific sensor location.	✓	
	C: A graphical report showing detections on a map of the park and areas within 2 miles of the park.	✓	
	D: A report showing detection classifications by ranger.	✓	
	The system should be developed in a way that would allow it to be easily reconfigured for other parks in the State of California.	✓	By adhereing to all the outlined standards in our book and class notes we ensured that our system program is easy to follow and add to for future improvments. The program has been unit tested, integrated tested and system tested.

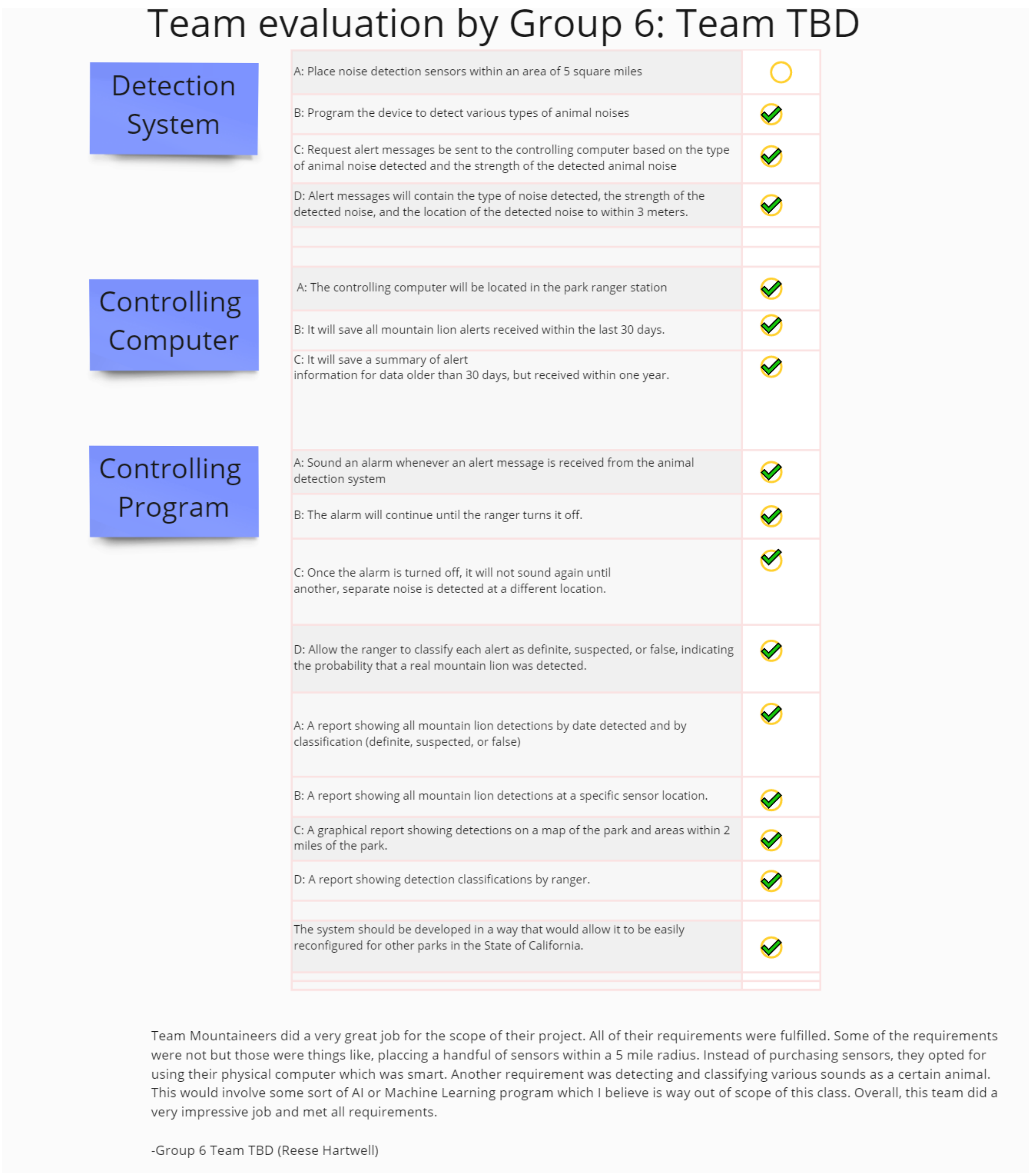
In general for the test Procedure and notes we took we can see the importance of testing in helping us integrate the programs together. Except for the initial detection, each other requirement had a prerequisite which we had to ensure it was met before it works. Our notes also show how it’s crucial to account for inconsistencies and only apply the checkmark when we have tested to make sure that those inconsistencies aren't present. In our notes we have divided the system into three parts and each part includes a couple of requirements that are essential for that part. This was a good practice specially because it helped us narrow down the requirements as essential functionality and use common sense to focus on detecting any defects for each of those functionalities. And while there have been times where we added to our notes odd issues that we faced such as the json saving an audio to be sent to the controlling computer and the issues that arise from that; overall everything was pretty straight forward.

Test Tool Development

We didn't design test tools for our project however we did use Postman automated API to test our node.js get and post code. We also Unit tested each of the python and node.js code individually. For the Postman automated API for example we did test our code to ensure that it does what it's supposed to do, however the automated API test didn't provide much feedback when things went south so it's main function was to test the code before we add an html/hbs to the program. For the system testing we tested how our Program would function when put together and made sure to fix any errors and update any method that prevented a smooth transition between the parts of the system. The testing is also mentioned in our checklist which narrows down all testing we did and which of the requirements passed its test cases.

Test Results Report

After receiving our evaluation from the other team our results came back as expected and that is mainly thanks to the numerous tests we put our system through to ensure its functionality. Below is the evaluation and feedback we received from the other team. The tests conducted by the team #6 were mostly system testing and ensuring that the program was functioning the way it was supposed to; team #6 chose to rate us by the same requirements that we outlined in our checklist as well as by personally using our system to ensure inconsistencies are not present in our system.



Team evaluation by Group 6: Team TBD

Detection System	A: Place noise detection sensors within an area of 5 square miles	○
	B: Program the device to detect various types of animal noises	✓
	C: Request alert messages be sent to the controlling computer based on the type of animal noise detected and the strength of the detected animal noise	✓
	D: Alert messages will contain the type of noise detected, the strength of the detected noise, and the location of the detected noise to within 3 meters.	✓
Controlling Computer	A: The controlling computer will be located in the park ranger station	✓
	B: It will save all mountain lion alerts received within the last 30 days.	✓
	C: It will save a summary of alert information for data older than 30 days, but received within one year.	✓
Controlling Program	A: Sound an alarm whenever an alert message is received from the animal detection system	✓
	B: The alarm will continue until the ranger turns it off.	✓
	C: Once the alarm is turned off, it will not sound again until another, separate noise is detected at a different location.	✓
	D: Allow the ranger to classify each alert as definite, suspected, or false, indicating the probability that a real mountain lion was detected.	✓
	A: A report showing all mountain lion detections by date detected and by classification (definite, suspected, or false)	✓
	B: A report showing all mountain lion detections at a specific sensor location.	✓
	C: A graphical report showing detections on a map of the park and areas within 2 miles of the park.	✓
	D: A report showing detection classifications by ranger.	✓
	The system should be developed in a way that would allow it to be easily reconfigured for other parks in the State of California.	✓

Team Mountaineers did a very great job for the scope of their project. All of their requirements were fulfilled. Some of the requirements were not but those were things like, placcing a handful of sensors within a 5 mile radius. Instead of purchasing sensors, they opted for using their physical computer which was smart. Another requirement was detecting and classifying various sounds as a certain animal. This would involve some sort of AI or Machine Learning program which I believe is way out of scope of this class. Overall, this team did a very impressive job and met all requirements.

-Group 6 Team TBD (Reese Hartwell)

Reflections

Comparison of Project Plans to Actual Performance

We followed most of our project plans, but there were exceptions like where we changed the architecture and database structure. We changed these features as it was necessary to achieve the most efficient code to run on the system's limited hardware. The final implementation of the project was demo-ed and performed as planned.

Project Lessons Learned

All the members on our team were very dependable and met the functional expectations on their assigned sections of the project. The main lesson learned was the necessity to stick to each of the individual tasks allotted time. However, we did manage to deliver the working prototype on time.

Summary of Customer Feedback on Prototype

Customer seemed happy with the presentation of the project, to quote one of the audience during our presentation, “I think you all did a really good job”.

Timesheet

Approach

We kept track of the time spent by each member weekly.

Date	Name	Hours
Week 1(8/23-8/25)		
	Halah	2
	Lewis	2
	June	2
	Total Hours this week	6
Week 2(8/30-9/1)		
	Halah	2
	Lewis	2
	June	2
	Total Hours this week	6
Week 3(9/6-9/8)		
	Halah	3
	Lewis	2
	June	2
	Total Hours this week	7
Week 4(9/13-9/15)		
	Halah	2
	Lewis	3
	June	2
	Total Hours this week	7
Week 5(9/20-9/22)		
	Halah	4
	Lewis	5

	June	4
	Total Hours this week	13
Week 6(9/27-9/29)		
	Halah	5
	Lewis	4
	June	4
	Total Hours this week	13
Week 7(10/4-10/6)		
	Halah	4
	Lewis	4
	June	5
	Total Hours this week	13
Week 8(10/10-10/13)		
	Halah	5
	Lewis	5
	June	5
	Total Hours this week	15
Week 9(10/18-10/20)		
	Halah	6
	Lewis	5
	June	5
	Total Hours this week	16
Week 10(10/25-10/27)		
	Halah	5
	Lewis	6
	June	5
	Total Hours this week	16
Week 11(11/1-11/3)		
	Halah	5

	Lewis	5
	June	6
	Total Hours this week	16
Week 12(11/8 -11/10)		
	Halah	3
	Lewis	4
	June	3
	Total Hours this week	10
Week 13(11/15-11/17)		
	Halah	4
	Lewis	3
	June	3
	Total Hours this week	10
Week 14(11/22-11/24)		
	Halah	3
	Lewis	3
	June	4
	Total Hours this week	10
Week 15(11/29-12/1		
	Halah	6
	Lewis	6
	June	6
	Total Hours this week	18
Final Week 16(12/6-12/8)		
	Halah	3
	Lewis	3
	June	3
	Total Hours this week	9
	Total Hours spent	Approximately 185 hrs

Deliverables

We have kept a detailed gitlab repository for the delivery of this project. The repository has a couple sub-repositories under it:

- Database Management Subsystem - This sub holds the code generated for controlling and managing a database.

- Detection Subsystem - This sub holds the code for the python noise detection program.

- GUI Subsystem - This sub holds the code for the react graphical user interface.

- DB-GUI-Integration - This sub holds the effort we did to integrate the DB and GUI systems. This code is also what we demo-ed for the class.

- MLDS Documentation - This sub was used to store links to each of the different pieces of documentation created with the project.