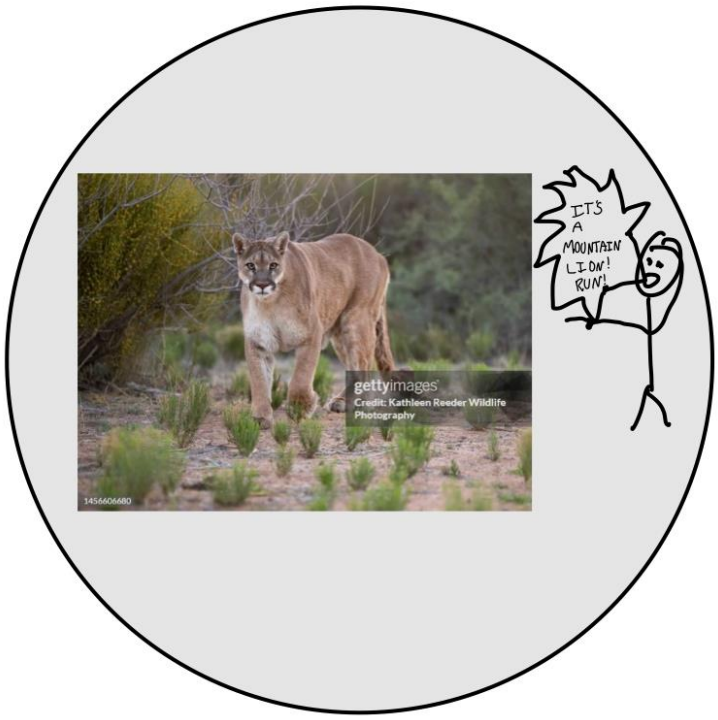
Mountain Lion Detecting System

Ian Hock, Lucas Coglianese, Mikah Martinez

September 27, 2024



Mountain Lion Detecting System

(Logo WIP)

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide a precise and comprehensive specification for the Mountain Lion Detecting System. The system will help park rangers with detecting, classifying, and reporting mountain lion activity based on noise detection technology. The system will be implemented on all park ranger stations in the San Diego Area and will help increase both efficiency and safety within the parks.

This document details the functional and non-functional requirements, user interactions, and system design constraints. It is going to help the development and implementation processes to make sure the final system meets the requirements of the stakeholders.

1.2 Scope

The Mountain Lion Detecting System is designed to monitor and alert park rangers of potential mountain lions nearby stations using noise detectors around the parks. Noise

detectors will be placed 5 miles apart from one another and will analyze noise detected. They will not only locate generally where the noise is coming from, but also categorize it (mountain lion, suspected mountain lion, or false detection), and notify the park rangers.

Key functionalities include:

- Noise detection and analysis
- Classification of the detected noise
- Alerts actively sent to the park ranger stations
- Reports generated based on mountain lion activity

This system will be a necessary tool to enhance the safety of San Diego's parks.

2. Overall Description

2.1 Product Environment

The Mountain Lion Detection System is apart of a broader initiative by the San Diego Parks and Recreation Department to increase safety around San Diego's parks. The system will be installed on a controlling computer in each of the park ranger stations around the San Diego region and they will each be connected to an array of noise detection sensors each spaced 5 miles apart. The system will utilize Wi-Fi to relay information between the sensors and the computer as well as send information to other park ranger stations.

This system will integrate into their current infrastructure of wildlife monitoring systems and enhance it by giving real-time alerts to nearby mountain lion threats. It will provide evolved safety measures utilizing noise measurement and classification.

2.2 Product Features

Key Features of the System include:

- Noise Detection and Analysis: Detect sounds in the environment and identify the level of severity.
- Classification: Determining the actual threat level.
- Alert System: Send alerts to ranger stations near the threat
- Real-time Monitoring: Constant monitoring of the environment, providing real-time data on the activity nearby
- **Report Generation**
    - Date and time of the detection
    - Classification of the detection

- Location of the detection
- A graphical map showing the mountain lion activity

2.3 User Characteristics

- **Primary Users**: Park rangers stationed at ranger stations
  - **Technical Proficiency**: Park rangers will vary in technical skills so the system must be intuitive and simple to use with minimal training
  - **Roles and Privileges**: Rangers will have administrative access allowing them to receive alerts and generate the detailed reports. Those who are not rangers and more generally those that are unauthorized will be restricted from accessing the system.

2.4 Operating Environment

The Mountain Lion Detecting System will operate on computers at each ranger station in the San Diego area. It will have access to Wi-Fi for communication as well as for updates. Additionally, the system must function well even with a varying Wi-Fi signal due to weather conditions.

If Wi-Fi communication fails, the system will still monitor sounds, but the alerts will be delayed until the connection is restored. If the Wi-Fi fails it should also try to notify the rangers that alerts will be down for the meantime.

2.6 Non-Technical Requirements

This subsection outlines the non-technical requirements to provide a clear understanding of the system's functions, how users will interact with it, and any constraints.

2.6.1 User Interactions and Functions

The system will be straightforward when it comes to interacting with it. The following list shows how the user can interact with the system:

- **Login and Access Control**
  - Park rangers will log in to the system using an admin account, and non-rangers will be denied access with an "Access Not Granted" message.
- **Alert Notifications**
  - When a mountain lion is detected, the system will send an alert to the controlling computer at the nearby ranger station. It will include information about where it is located as well as the classification of the detection (definite, suspected, or false).
- **Report Generation**

- Rangers will be able to generate reports of all activity in a radius around a specified location, a graphical report showing detection on a map with a 5-mile radius around a park, and a map with the San Diego area and where mountain lions have been detected. These will all include the date and time of each detection as well as the classification.

2.6.2 Hardware Context

The system will operate on a standard computer and should not require heavy processing power. The system is described by the following:

- **Controlling Computer**
  - Each ranger station will be equipped with one controlling computer that handles the noise processing, alerts, and report generation
- **Noise Detection Sensors**
  - These sensors will be provided by the department, but the controlling computer will get the noise data and must process it. It will be sent as a .WAV file.
- **Wi-Fi Infrastructure**
  - A reliable Wi-Fi connection will be required to allow transmission of sound data and alerts. If Wi-Fi fails, data will be only stored locally and processed once the connection is restored.

2.6.3 UI (User Interface) Requirements

The user interface (UI) must be simple, intuitive, and user-friendly to accommodate the range of technical skills among the rangers. The system will feature:

- **Clear, Easy-to-Read Alerts**
  - The UI will present clear notifications on the mountain lion detections. They must be easily distinguishable with the location and classification displayed clearly.
- **Map-Based Visualization**
  - The UI will also include maps showing the nearby area with points where mountain lions are detected. These must be easy to read and have plenty of coverage.
- **Report Display**
  - Reports must be easily generated through the interface and easy to view and analyze. They will be printed and displayed digitally

2.6.4 Software and Development Constraints

The following constraints guide the development of and implementation of the system:

- **Programming Language**
  - The system should be developed using languages compatible with the existing infrastructure at the ranger stations. Python and Java are recommended for the back-end due to their data processing powers, while HTML/CSS/JavaScript can be used for front-end UI.
- **Operating System**
  - The system must be compatible with Windows as that is what is compatible with the current infrastructure
- **Performance Constraints**
  - The system must respond to the detected noises within 2-3 seconds as any delay would compromise the safety of the nearby area.
- **Scalability**
  - The system should also be able to accommodate any new ranger stations or if there is an increased number of detectors in the future.

2.6.5 Constraints and Limitations

There are a few constraints and limitations that must be accounted for:

- **Dependency on Wi-Fi**
  - The system heavily relies on a Wi-Fi connection for the real-time alerts. The system will have limited functionality while Wi-Fi is down, but as soon as the connection is restored the alerts must be quick to return as well.
- **Noise Differentiation**
  - While the system is designed for mountain lions, it may get confused will other large animals and this must be adapted as more data is processed. This means the system should also be able to determine other animals in the future as well.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 Noise Detection

The system will have noise detection sensors distributed around the parks 5 miles apart. It will then use the signatures of the animals to identify the type of animal (specifically focusing on mountain lions).

3.1.2 Classification

The noises that are detected will be labeled with either definite, suspected, or false. This will allow the rangers to be wary of nearby threats. This will eventually include the ability to identify other animals other than mountain lions.

3.1.3 Alert Notification

Alerts will be generated as quickly as possible while being no longer than 3 seconds. It must state clearly the location as well as the classification it falls under. They will only be displayed on the closest controlling computer, but it can be sent to other computers if needed.

3.1.4 User Login

As stated previously, park rangers will have administrative logins that will allow them to make reports and use the functions of the computer. Those who are not authorized (non-rangers) will not be allowed to access the system.

3.1.5 Reports

The system will have a few different types of reports: recent alerts of a specified area, graphical map with recent detections around the controlling computer, and a general map of the San Diego area that has all alerts in some time frame. These reports will be printed as requested by the ranger

3.2 Non-Functional Requirements

3.2.1 Usability

The system will have a large focus on being clean, clear, intuitive, and easy to use. Park rangers should not require much training, and the UI will be easily read by the public.

3.2.2 Performance

The system should classify the noise no longer than 3 seconds after detection. The actual alert must show up on the respective controlling computer within 5 seconds after classification.

3.2.3 Reliability

The system must be operational 24/7, with minimal downtime only during updates. The classification and noise detection system should be accurate at least 90% of the time to minimize false negatives and positive.

3.2.4 Security

Only authorized rangers should have access to the system. This means that the system should be password protected requiring a login.

3.2.6 Language

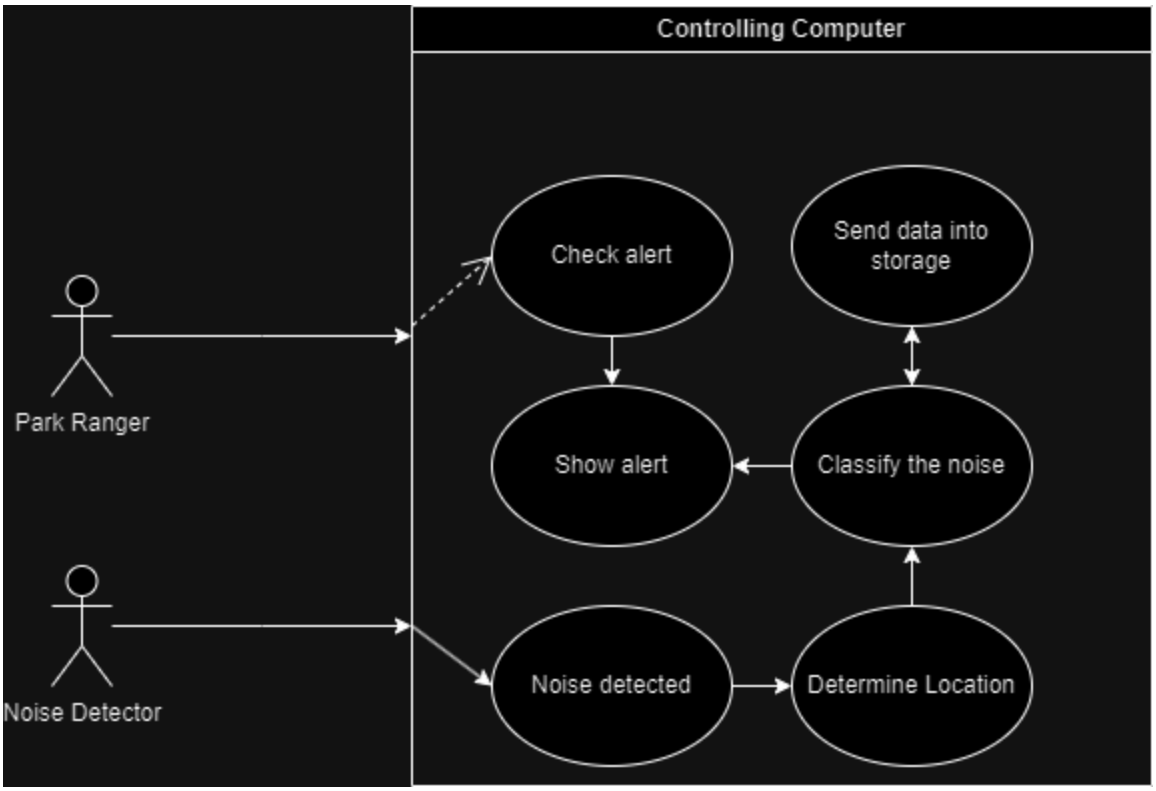
The system will be available in English only for now. Other languages may be added in the future.

4. System Models

4.1 Use Case Diagram

4.1.1 Normal Flow – Noise Detected

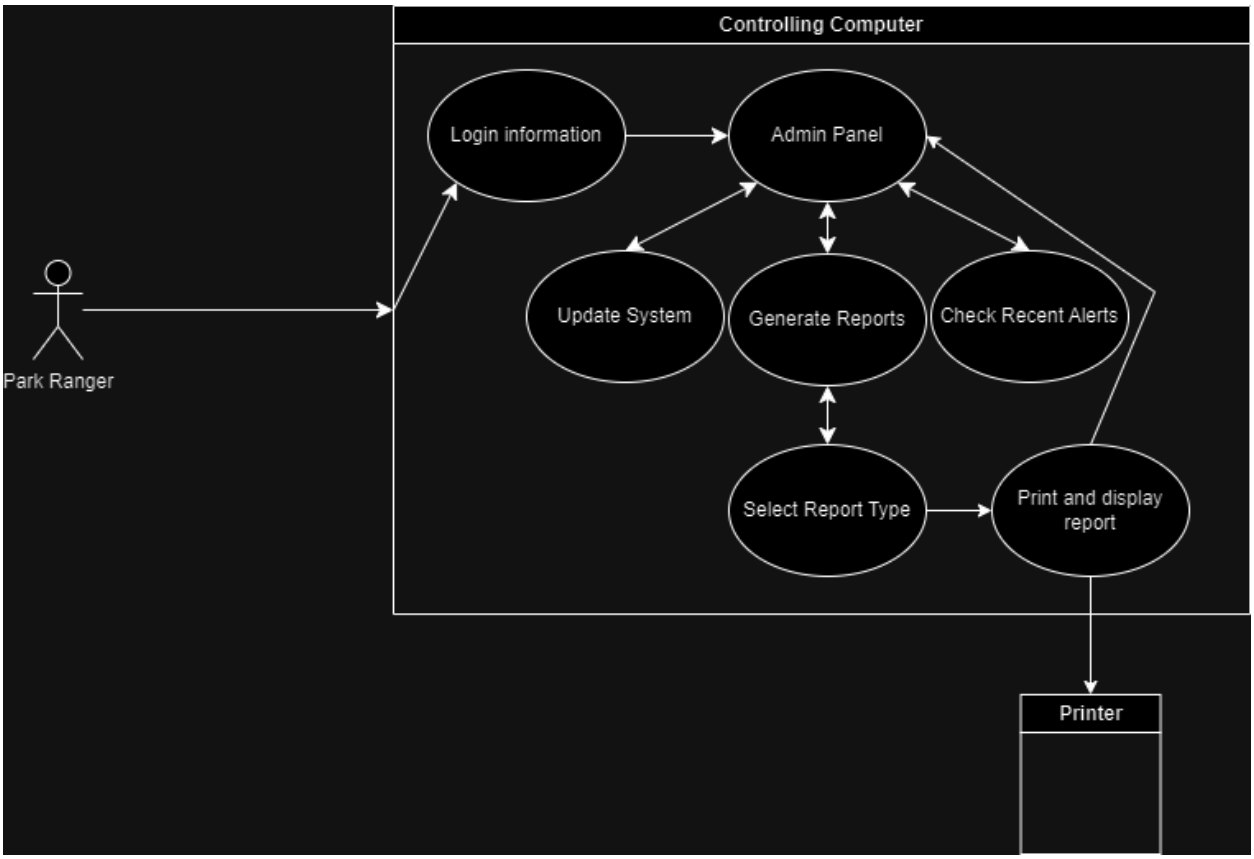
The first use case diagram represents when a noise is detected:



The noise detectors are practically an actor in this situation as they provide the main input. The controlling computer than realizes a noise is detected and begins to process the noise. It first determines its approximate location due to the strength of the noise and the location of the noise sensor that heard it. It then classifies the noise into the three categories and sends the data to the storage as well as the alert. This flow also can take input for the park ranger as they can check the alert and this will show the data of the noise.

4.1.2 Normal Flow – Park Ranger

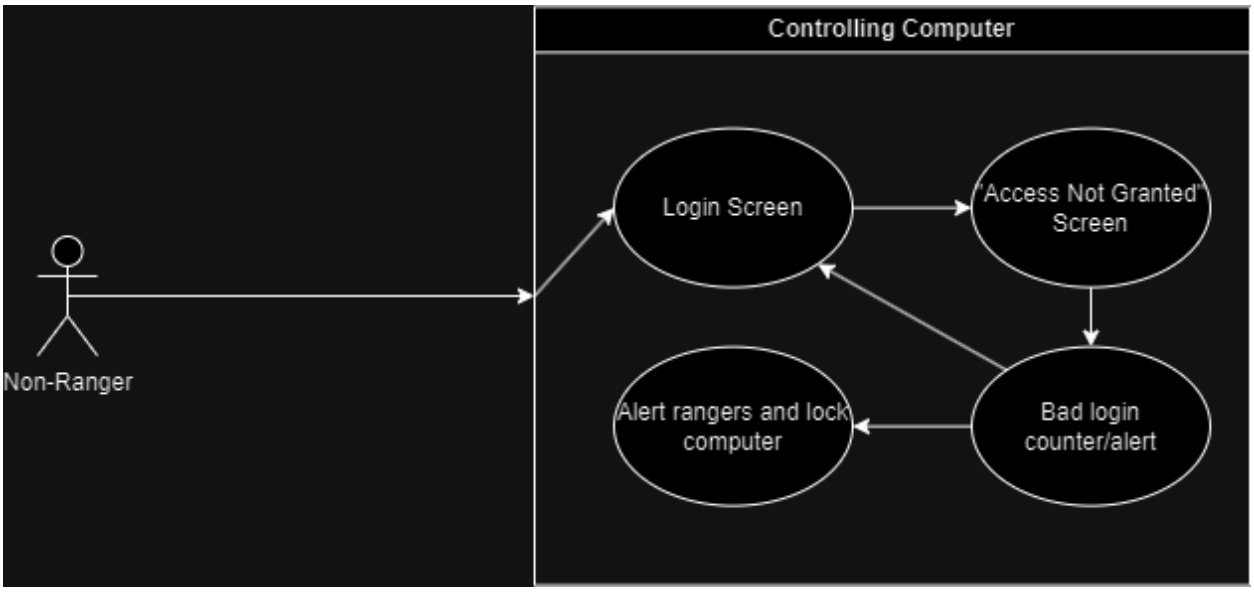
The next diagram shows the other situation in which the park ranger accesses the system:



This diagram starts with the ranger using their credentials to login to the system. This then brings up the administrative panel UI which allows the ranger to access the whole system. This will allow them to either update the system, generate a report, or check the recent alerts, all being able to go back to the admin panel (home page). If the user decides to generate a report, they will get prompted to select the type of report desired. They will have the 3 options stated above and after selected it will print the report and go back to the home page.

4.1.3 Exceptional Flow – Unauthorized User

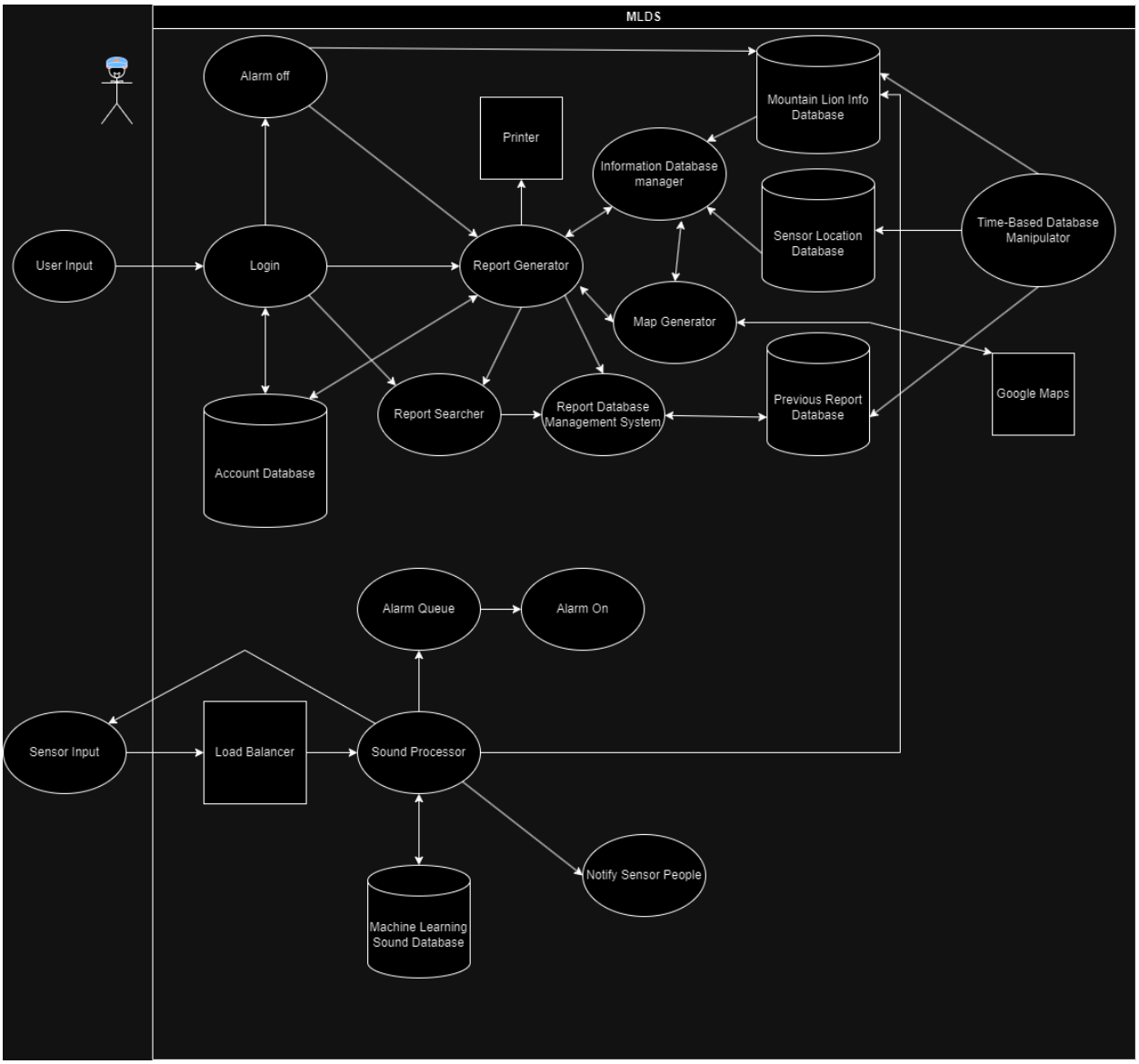
There will be a few exceptions, but the most common one will be an unauthorized user. The following diagram describes that situation:



The general exception of an unauthorized user will be handled quickly as valid credentials will be required to login to the system. As seen by the diagram the non-ranger will attempt to login, but their credentials will not be recognized as valid. This will send them into the “Access Not Granted” Screen as well as increment the number of bad logins. After 5 bad logins in 5 minutes, it will lock the controlling computer and alert other park rangers.

5. System Architecture Overview

5.1 System Architecture Diagram



5.1.1 System Architecture Diagram – Component Descriptions

- **User Input**: Interacts with the Login component
- **Login**: Interacts with the Account Database to validate login and store its information, then based on what is needed from the user, connects to the Alarm off, Report Searcher, or Report Generator
- **Alarm On and Alarm Off**: Controls alarm state, which is connected to the **Alarm Queue**

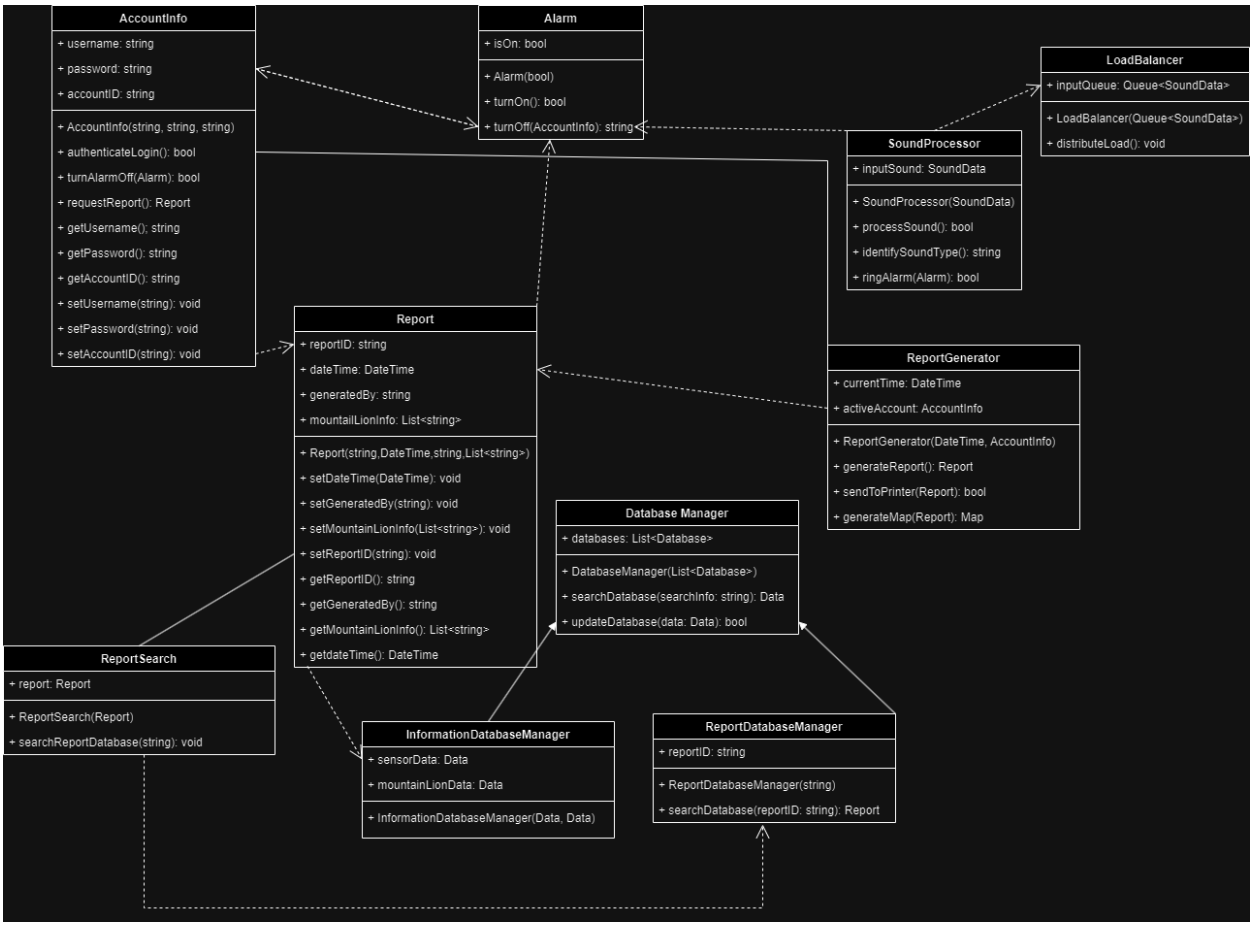
- **Account Database:** Holds all account information for every user which is used by the **Login** component and **Report Generator**
- **Report Generator:** Creates reports based on the required information which connects to:
    - **Printer:** An external system used for printing the reports
    - **Report Searcher:** For using data from previous reports
    - **Map Generator:** Creates maps for specified report being generated
    - **Information Database Manager:** Gathers additional data from the **Mountain Lion Info Database** and **Sensor Location Database**
    - **Account Database:** Stores account information which is used to assign users to their generated reports
    - **Report Database Management System:** Filters reports based on (user, date, area, etc.)
- **Mountain Lion Info Database:** Stores information held on mountain lions, such as noises created and times in which they were created, which is used by the **Sound Processor**
- **Sensor Location Database:** Keeps track of the locations in which the sensors are placed
- **Previous Report Database**: Stores recent reports and connects to the **Report Database Management System** for report generation and search uses
- **Map Generator**: Generates maps using data from the **Sensor Location Database** and **Google Maps** integration
- **Time-Based Database Manipulator**: Manages time-sensitive data in the **Sensor Location Database** and **Previous Report Database** for generating accurate reports
- **Google Maps**: External service used for location-based mapping, utilized by the **Map Generator**
- **Alarm Queue**: Receives inputs from the **Sound Processor** and determines when to trigger the **Alarm on** state (e.g. if there is already an alarm sounding, it will play another when turned off)
- **Sensor Input**: Provides sensor data to the **Load Balancer**
- **Load Balancer**: Helps with multiple inputs and distributes sensor data to the **Sound Processor** for further analysis
- **Sound Processor**: Processes the incoming sensor data using the **Machine Learning Sound Database** to identify relevant sounds – mountain lion noises. It connects to the **Alarm Queue** if valid noise and notifies sensor personnel via **Notify Sensor People** if not valid

- **Machine Learning Sound Database**: Contains trained sound models for detecting mountain lion sounds, used by the **Sound Processor**
- **Notify Sensor People**: Communicates with relevant personnel to notify them any issues with hardware or software from sensor

5.1.2 System Architecture Diagram – Key Dependencies

- **Login** depends on the **Account Database** for user validation
- **Report Generator** is a central component dependent on the **Information Database Manager**, **Map Generator**, **Report Searcher**, and external systems like the **Printer** and **Google Maps**
- **Sound Processor** relies on the **Machine Learning Sound Database** for detection, and its output influences the **Alarm Queue** and notifications
- **Databases (Mountain Lion Info, Sensor Location, Previous Report)** are critical for various components, primarily managed by the **Information Database Manager** and **Time-Based Database Manipulator**

5.2 UML Diagram



5.2.1 UML Diagram – Component Description

We have several classes that will be used in the mountain lion description. Most of these classes will interact with one another.

- **AccountInfo**: this class will be used to store data for an account in one object that can then be stored in the account database. AccountInfo will have three strings that are the username, password, and userID. Most of the operations consist of getters and setters, but it also has an operation to authenticate a user as well as one to search for reports. It also has an operation that interacts with the alarm, being able to only turn it off.
- **Report**: this class will be used to store data on each report that is generated. This means that it will be used by not only the report generator, but also the report search. Report has four attributes that consist of the following:
    - **ReportID**: a string that will have a unique identifier for each report
    - **dateTime**: a DateTime object that holds the date and time that the report was made
    - **generatedBy**: an AccountInfo that keeps track of who generated the report
    - **mountainLionInfo**: hold a list of data about the mountain lion that was detected. It will hold the classification and location of the mountain lion.

  It also has just getters and setters for the operations for these attributes.

- **ReportSearch**: A class used to interact with the database and look through the previously produced report. The ReportSeach class will use a given report and use the operation searchReportDatabase and send the reportID to the ReportDatabaseManager which then searches the report database for that report.
- **InformationDatabaseManager**: this class is made to manage the data within the mountain lion information database (classification, which sensor detected it, and the ranger who turned the alarm off) as well as the sensor location database. It uses this to be able to generate reports and locate where the mountain lion was detected.
- **SoundProcessor**: SoundProcessor takes in sound data such as a .WAV or .MP3 file and uses two of its operations (processSound and identifySoundType) to classify the sound into the three types (definite, suspected, false). There is another operation that uses an Alarm object as the parameter to sound off the alarm.
- **Alarm**: alarm is a simple class that has two operations, turnOn and turnoff. These two operations alter the isOn attribute which states whether the alarm is ringing or not.

- **DatabaseManager**: this class is a generic class that is able to modify and interact with databases. It has a list of databases that it can interact with using its operations. The two operations are searching the database and update the database. Searching the database returns a piece of data while update takes data and alters the database.
- **ReportGenerator**: this class takes the current time and the active user as the attributes as well as uses the report class to generate a report using three operations. It can generate a report, generate a map, or send a report to the printer. The second two reports use a report to print the report if desired or generate a map to visualize the whole San Diego area.
- **LoadBalancer**: this class generally will be used by the sound processor to retrieve data from the sensors. LoadBalancer takes in all the sounds for sensors and sends them to different sound processors to make sure everything runs smoothly.
- **ReportDatabaseManager**: this class is a database manager that uses a reportID to search the report database and return the full report. This can then be used to print the report or generate a map of the report.

For assignment 3, we updated our UML class diagram by altering some of the return types. Specifically, we changed the Alarm to return a Boolean if it successfully turned the alarm on. We also added constructors for each class to ensure that we can make objects and test each part of the system.

6. Data Management Strategy

6.1 Database Specification

We have a total of five databases as seen in the software architecture diagram. The five databases will all utilize SQL and are as follows: Machine learning sound, account, previous report, sensor location, and mountain lion information database.

- **Machine learning Database:**
    - Using SQL this database will store the sound files and classification of each sound. This will be used by the sound processor to learn and adapt as it gets more data about mountain lion sounds. It will use machine learning to learn and develop a better algorithm for determining the classification of a sound.
    - Below is an example of the table used in the database:

Sound ID (int)	Sound File (File)	Classification (String)
1	2024-10-311200.MP3	“Definite”

2	2024-10-311426.MP3	“Suspected”

- **Account Database:**
  - This database uses SQL to hold all information for each ranger’s account. The Mountain Lion Information database will use references from this database for specified actions.
  - Below is an example of the table used in the database:

ID (int)	Username (String)	Password (String)	Name (String)	Position (String)
000001	“mikahmartini”	“Password”	“Mikah Martinez”	“Ranger”
000002	“bobtheminion”	“I<3GRU”	“Bob Gru”	“King”

- **Sensor Location Database:**
  - This database will use SQL to store the locations of each sound sensor in latitude and longitude doubles, with each sensor having its own unique ID. This will allow the location of a sound to be recorded with each report.
  - Below is an example of the table used in the database:

ID (int)	Latitude (double)	Longitude (double)
1	32.715736	-117.161087
2	32.825125	-116.251562

- **Mountain Lion Info Database**
  - This database stores information in SQL about each mountain lion detection that occurs. It includes whether it’s a definite or suspected mountain lion, as well as the user ID of the person who turned off the alarm.
  - Below is an example of the table used in the database:

ID (int)	Sensor ID (int)	Sound file (File)	Classification (String)	User ID (who turned it off) (int)
0001	2	/0001.wav	“Definite”	0215
0002	8	/0002.wav	“Suspected”	0184

- **Previous Report Database**

- This database utilizes SQL to store and retrieve data containing all of the information needed for the specified report including the date the report was created, who created it, and the pdf file of the report.
- Below is an example of the table used in the database:

ID (int)	Date (String)	Name (String)	Report (File)
0001	“2024-09-07”	“Mikah Martinez”	2024-09-07-1200.PDF
0002	"2024-10-10”	“Bob Gru”	2024-10-10-1200.PDF

6.2 Database Reasoning

We have five databases in our system, and each corresponds to the necessary information for specific cases. We could have used a NoSQL database such as Redis and had IDs and Classifications as keys with their respective values, but the relational database layout makes more sense and is easier to work with for the purposes of our system. The data is split up into different sections for each system. For example, we have the machine learning database for our specific sound processing part of the system. Mountain lion information is for the alarm and generating reports, previous reports for generating reports about the previous reports that were made, and sensor location is used for storing the exact coordinates of every sensor. For the sensor location database, we had the option to use key-value pairs having the ID as the key and the latitude and longitude as the value, but we decided it works better to have a table that can work with the mountain lion information table as well. This also works with generating reports as everything will be table-based and can be generalized. The utilization of table-based databases for all of our databases means they are able to be manipulated easily, and functions can be made generally.

We decided to keep our existing software architecture diagram, as we feel that we’ve already partitioned our data into databases in an acceptable way. We already separated things into categories and have databases that work with one another as seen by the mountain lion database that uses the location of sound sensors.

7. Development Plan

7.1 Task Partitioning

The mountain lion detection system requires multiple steps to set up, as well as different development tasks. This section outlines how these tasks and steps will be divided between our teams.

Development Tasks

Development of the control system will be a team effort. However, different team members will manage different parts of the software.

- **UI/UX:** Mikah will manage and lead the development of the UI for park rangers to interact with the system.
- **Networking/Sensor Relays:** Ian will manage developing and maintaining the networking system between the sensors and the control computer.
- **Database/Backend:** Lucas will develop and manage the database system as well as the backend systems that interact with each database.

Setup Tasks

To set up the system in a new park, sensors must be properly placed and connected to the control system. The team will find the optimal locations for the sensors and set them up there. Additionally, we will all take part in the installation and connection of the control system, to ensure that each of our development areas listed above is working properly.

7.2 Timeline

Different phases of this project will take time to develop, install, and set up. As such, here is an outline of the ETA for each phase of the project:

- **Development: 5 months**
    - This includes development of the software for the control system as well as setting up databases and the networking system for connecting to the sensors.
- **Installation: 1 month**
    - Installation entails getting the proper approval from the San Diego Parks and Recreation department as well as installing sensors and connecting them wirelessly to the control system.

- **Setup and training: 1 month**
    - Setup includes setting up and securing the control system in a ranger station. Park rangers will also have to be trained how to use the control software and how to respond to an alert.

Total estimated time of arrival: 7 months

7.3 Tests

The two test sets we are going to check are going to surround the alarm ringing based on a sound being inputted and someone trying to print a report.

7.3.1 Alarm System

- Unit Test: Alarm.turnOn()
    - Alarm a = new Alarm();
      bool turnedOn = a.turnOn();
      if(!turnedOn) return FAIL;
      else return PASS;
    - In this test we take an alarm object and use the turnOn method to turn the alarm on. If it successfully turns on it returns true, otherwise it returns false. If the method returns false, then the test fails and so we return FAIL.
- Functional Tests: SoundProcessor.ringAlarm(Alarm)
    - Test 1:
      SoundData sound = new SoundData(“./tests/mountainlion.wav”);
      Alarm alarm = new Alarm();
      SoundProcessor processor = new SoundProcessor(sound);
      bool success = processor.ringAlarm(alarm);
      if (!success) return FAIL;
      else return PASS;
    - Test 2:
      SoundData sound = new SoundData(“./tests/mountainlion.pdf”);
      Alarm alarm = new Alarm();
      try{
      SoundProcessor processor = new SoundProcessor(sound);
      }
      catch FileIOException{
      Return PASS;
      }
      Return FAIL;

    - The first test ensures that the SoundProcessor object is correctly able to interact with the SoundData and Alarm objects to ring an alarm. If the ringAlarm method returns true (indicating successful alarm activation), then the test passes. The second test tests if the sound processor when being created detects the wrong file extension. If the sound processor constructor determines that the extension of SoundData is not .WAV or .MP3 then it will return a FileIOException. If this occurs, then the test was passed as we sent in a .PDF file. Otherwise, it failed miserably.
- System Test: Alarm sounds when audio is processed and classified as ML
    - First, a mountain lion growls/roars with a noise detector which then records the noise as an .MP3 and sends that to the system. The system then uses the sound processor class to process the sound and return a classification. When the sound processor determines that the sound is a mountain lion it should ring the alarm waiting for a park ranger to deactivate it. If it does not ring it and it was a mountain lion, then the test has failed.

7.3.2 Report System

- Unit Test: getReportID
    - Test 1:
      Report report = new Report();
      report.setReportID(“ABC123”);
      if(report.getReportID() != “ABC123”) {
      Return FAIL;
      }
      else{
      Return PASS;
      }
    - Test 2:
      Report report = new Report();
      try {
      report.setReportID(“”);
      }
      catch EmptyStringException {
      return PASS
      }
      return FAIL
    - The first test checks the getReportID method and checks if the reportID is the same as the reportID created for the report. If it is the same then it passes, if

not then it fails. The second test checks if we can set a reportID to an empty string. This should raise an error as we do not want empty strings as reportID's. So if it catches an error, it returns PASS, otherwise it returns FAIL.

- Functional Test: sendToPrinter(Report): bool
    - AccountInfo account = new AccountInfo("testuser", "testpass", "1");
      ReportGenerator generator = new ReportGenerator(DateTime.now(), account);
      Report report = generator.generateReport();
      bool success = generator.sendToPrinter(report);
      if (!success) return FAIL;
      else return PASS;

    - This test checks that the ReportGenerator is successfully able to generate a report given account info, as well as is able to send that report to a printer. If the send is successful, the method returns true, and so the test passes.
- System Test: Print Report
    - First, a user interacts with the UI and selects a report they want to make. Next, the user clicks the print button in the report UI, triggering the ReportGenerator class sendToPrinter method. After this method is invoked, we ensure that the printer physically prints out the report. If it did not print then this system test failed.

8. Appendices

8.1 Glossary of Terms

- **Detection Radius**
    - 5-mile radius around noise detection sensors
- **Admin Panel**
    - The home screen of the controlling computer after a valid login that allows the user to see alerts and generate reports
- **Controlling Computer**
    - The central computer located at each park ranger station that processes the noise data and sends alerts
- **Lockout**
    - A security measure triggered after multiple failed login attempts, which blocks further access to the system temporarily while alerting authorized users.

8.2 Major Risks

The main risks associated with the Mountain Lion Detection System are primarily related to its dependency on Wi-Fi for real-time communication and data transmission. The system is designed to run reliably, but several factors can impact performance:

1. **Wi-Fi Reliability**
    - **Risk**
        - The system relies on a stable Wi-Fi connection to transmit data and send alerts. Environmental conditions could reduce Wi-Fi reliability causing delayed alerts or failed data transmission
    - **Mitigation**
        - If the Wi-Fi signal ever weakens or fails, the system will attempt to operate but in a degraded state. However, alerts may be delayed, but it will make sure to notify park rangers of the unreliable connection.
2. **Reduced Data Accuracy**
    - **Risk**
        - Spotty Wi-Fi could lead to a decrease in reliable data transmission and accuracy
    - **Mitigation**
        - To counteract this risk, the system will include protocols to identify if any incomplete data was received. Alerts will be adapted to make sure rangers know to be wary until the full data is received.
3. **Sensor Malfunction**
    - **Risk**
        - Environmental conditions may cause damage to the noise sensors
    - **Mitigation**
        - Routine diagnostics will be built into the system to check the health of the sensors and test them consistently. If there is a malfunctioning sensor the nearby ranger station will be alerted.
4. **False Positives/Negatives**
    - **Risk**
        - The system may generate false positives (alerting on a non-mountain lion noise) or false negative (failing to detect an actual mountain lion). This can be due to noise interference or calibration issues.
    - **Mitigation**
        - The system will implement machine learning and regular updates to increase its data of animal noises and fine-tune its noise detection

and classification algorithms. Rangers will also be made aware of the possibility of false positive and negatives.

8.3 System Testing Plan

A system testing plan will be developed to ensure the quality of the system is of the highest level. It will validate the functionality, performance, security, and user experience of the system. Every aspect of the system, such as the noise detection and classification system, will be tested thoroughly to establish reliability.