Mountain Lion Detection System Technical Report

Noah Garcia
December 1, 2022

Table of Contents

—User Manual [1.0]—

Section 1.1 - System Introduction and Overview

Hello, and welcome to our Project Report! Recently, we have met with a client who has asked us to help them design and build a software system of their choosing by having them specify all the requirements that they would like the system to possess. The purpose and goal of this software system is to gather information on the most frequent sightings of mountain lions for the San Diego County Parks and Recreation Department to analyze. The reason this software system is needed is because the presence of mountain lions near recreational parks in the San Diego County area has led to some safety concerns for tourists and visitors, as well as concern for the safety of the mountain lions themselves, since they are an important part of native wildlife in Southern California's ecosystem. Therefore, this software system will provide the adequate resources to detect, analyze, and store information on the presence of mountain lions who may roam around parks, recreational centers, and other outdoor areas in San Diego. Those who will be most affected by the use of this system are the park rangers, park managers, and locals or tourists who visit these recreational parks and areas for their leisure. This document has been organized in such a manner that the information pertaining to the software system is neatly presented for the client, the user, and the stakeholders, so that they may understand the underlying components and functions of the system. In addition to this brief introduction, this document includes information about the user requirements, the system requirements (both functional and non-functional), as well as any risks, constraints, assumptions, or future potential changes about the system. Use-Case diagrams have also been included in the description of functional requirements for transparency on how the system will work and flow visually.

Section 1.2 - User Requirements

The company, Animals-R-Here, will be providing the hardware on which our designed software system will run. They have also informed us that the computer they have provided is manufactured by Dell, however the hardware is a bit older and only accepts Java and Python code in order to execute systems. This presents us with a coding constraint, therefore, we will build our software using Java language in order for the system to run smoothly on the hardware without any compiling problems. The client has also informed us that there is no budget, however we are constrained to a hardware system that is older than what we may be used to, so we must keep the software as simple and efficient as possible while also making sure it executes all the desired functions for the client. Furthermore, the client has requested that the program also contain a login system that is only accessible by the park rangers or managers, in order to keep all the information stored on the computer as secure and private as possible. This controlling computer will be located inside of the park ranger station where the users will be able to view, add, or delete information once they login. The software system will run in conjunction with a detection system previously developed by the Animals-R-Here company, which involves setting up several noise sensors over a concentrated area in order to detect the presence of certain

animals; in this case we are detecting the presence of mountain lions. When a suspected noise has been picked up by a sensor that it perceives as a mountain lion, it will automatically send alert messages to the controlling computer in order for the information to be analyzed by the user. Most of the time, those on the receiving end of the alert message will be a park ranger, but may also be a park manager as well. Once a new alert message has been sent by the detection system and received by the controlling computer, the user will have to manually turn off the alarm that is sounded using an off-switch; the alarm was requested by the client to enforce a sense of urgency in the workplace. The user will then be able to analyze whether or not the noise belongs to a mountain lion based on the type of noise detected and the level of its strength. Unfortunately, one major constraint of this system will require us to have the user analyze the source of the noise, record the data, and store it on the computer manually. This flow might seem inefficient or error-prone due to human mistakes, however, the hardware on which our system will run doesn't allow for us to program the system in order for the computer to analyze the noise detections directly. Thus, the task of analyzing will have to be done by the user who is present at the time. The user will also be able to view and modify all information stored on the computer's hard disk drive at any given time through a series of steps or actions described in detail later on.

Section 1.3 - Description of Features and Functions

There will be many prominent features and functions included in our system in order to help our clients complete their assigned tasks. Features will include, but are not limited to, our system displaying speed, accuracy, diligence, reliability, memory, logical, versatility, automation, consistency, and remembrance power. We pride ourselves on making sure that our system is well-presented and functions to the best of its ability in both quality and quantity. Functions will include, but are not limited to, a login process for the user to access the system in a secure manner, a homepage where the user will be able to decide what action they want to take once they are inside the system, and various directories that hold different types of relevant information. These directories are included for the purpose of structurally organizing all of the data that has been collected, stored, and compiled by not only the system, but the currently-active park rangers themselves. They contain various functions that allow the user to get the work done as efficiently as possible on their own, as requested by our client. The directories will store important information such as the history of alerts that have gone off in order to get the attention of staff that is working at the time, data collected and stored by certain operating sensors in parks and recreational centers, satellite-accurate maps of those parks and recreation centers that store coordinates of various wild animal sightings, a catalog that grabs every single file of information available on the computer and stores it into a structured and organized list of data files, and the information of every currently-active park ranger working with the system for transparency.

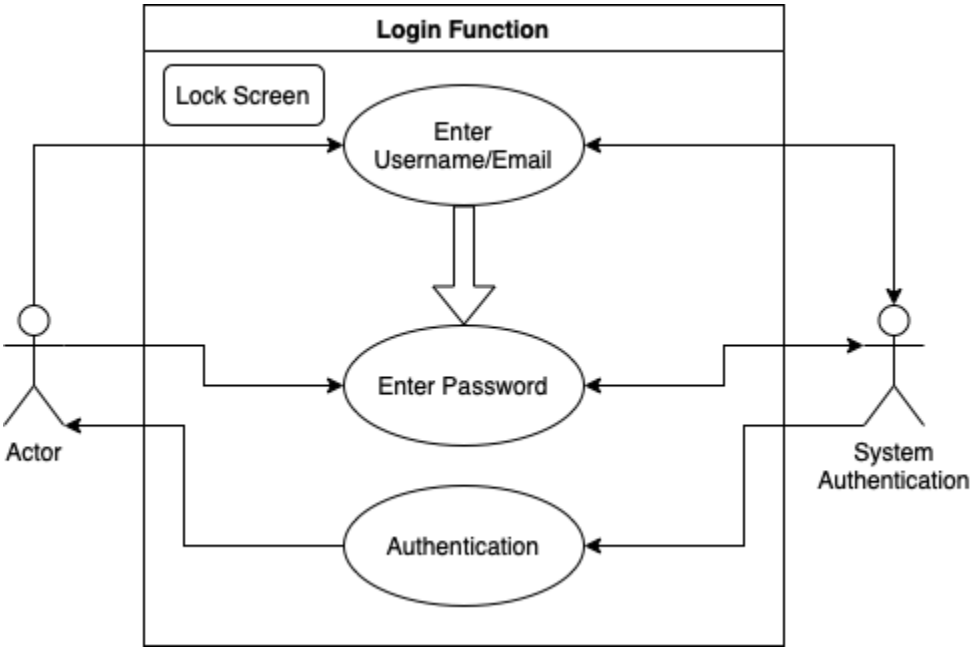
—Design Documents [2.0]—

Section 2.1 - System Requirements

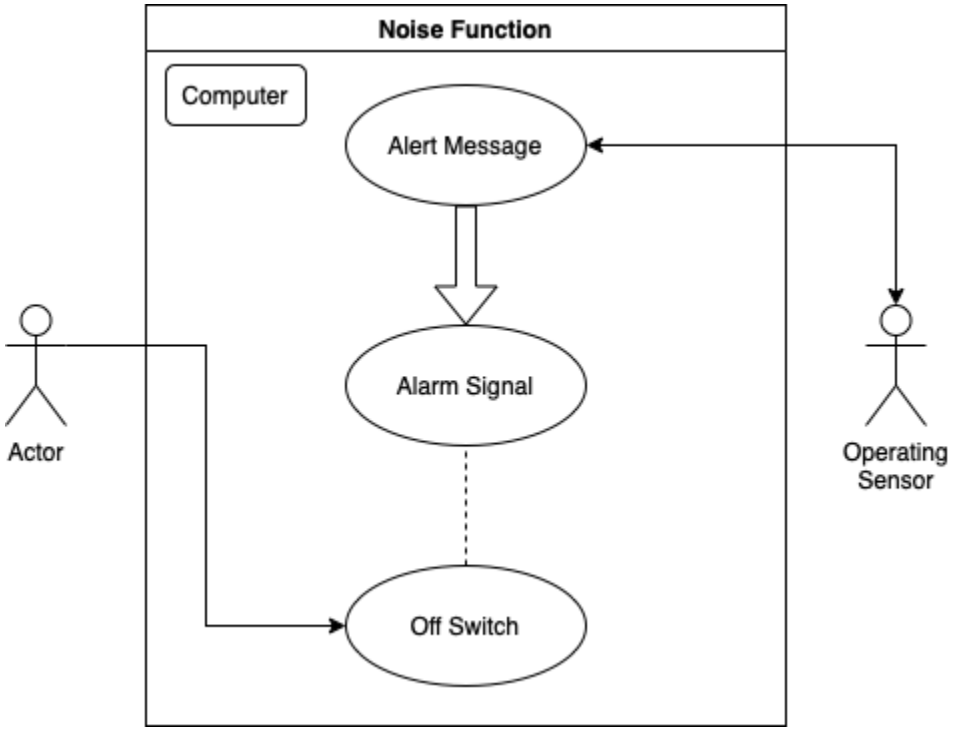
Section 2.1a - Functional Requirements

Below are the functional requirements that our system possesses. These requirements are about the system's functions and what it does for the user. We included Use-Case diagrams for the purpose of showing how the system's functions will flow in a visual manner, which is better than just reading a paragraph. Using one diagram would become very messy and confusing for the reader, so we decided to separate each function into its own diagram in order to avoid any confusion and improve legibility.

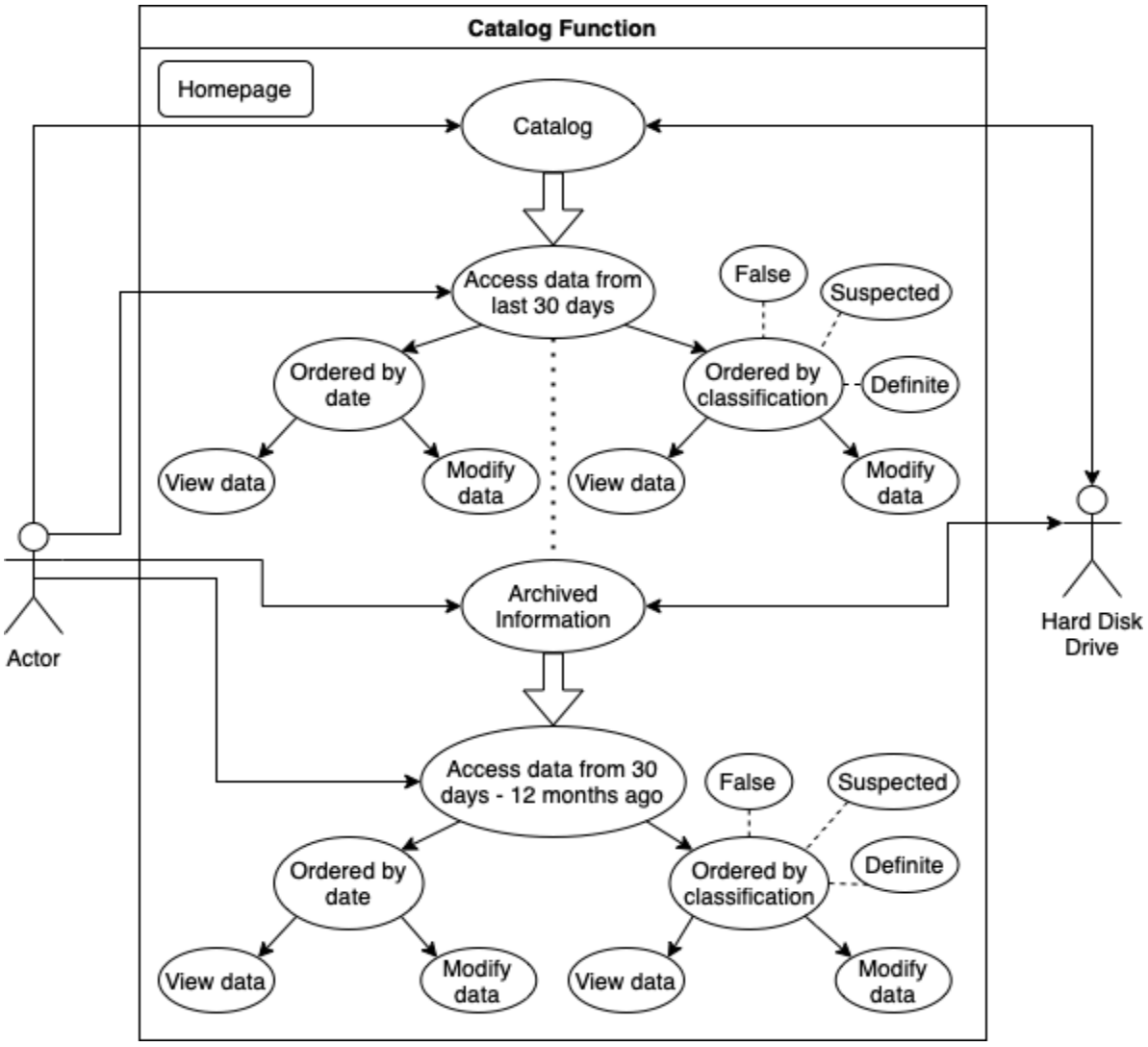
- The secure login system will consist of requiring the user to input their assigned username and password in order to gain access to the computer's contents. It is worth noting that each and every staff member will have their own personal username to enforce privacy and discretion, as requested by the client. Staff will also be able to use their work emails to login as well. Once the user enters their username or email on the lock screen, it will be run through the system authentication service to check validity. Once this process is finished and the system gives clearance, the user will enter their password so that it may also be verified by the system. Once the username and password have both been cleared by the system, the user will be granted access to the computer.



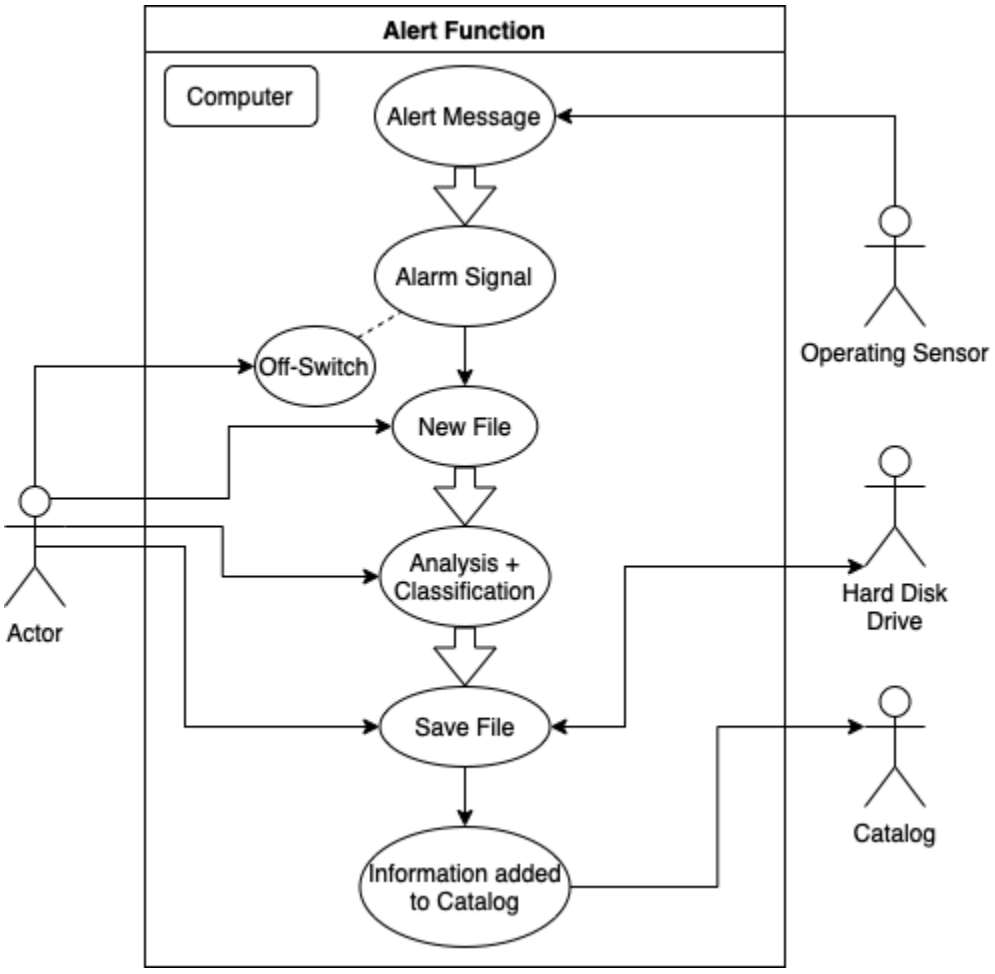
- The noise detectors that are set up in the parks and recreational areas are programmed to detect various types of animal noises to increase their accuracy. Once a noise has been detected by the sensor that it perceives as a mountain lion, it will automatically send an alert message to the controlling computer located inside of the park ranger station. When the computer receives the alert message from the animal detection system, it will be prompted to sound an alarm signal in order to notify the park rangers that the sensors have picked up a noise, and that noise might belong to a mountain lion. Once the park ranger has received this alert message, the alarm will continue to sound until they manually turn it off via an off switch. Once the alarm has been manually turned off by the park ranger, it will not go off again until a new noise has been detected by a different sensor, after which a new alert message will be sent out to the controlling computer.



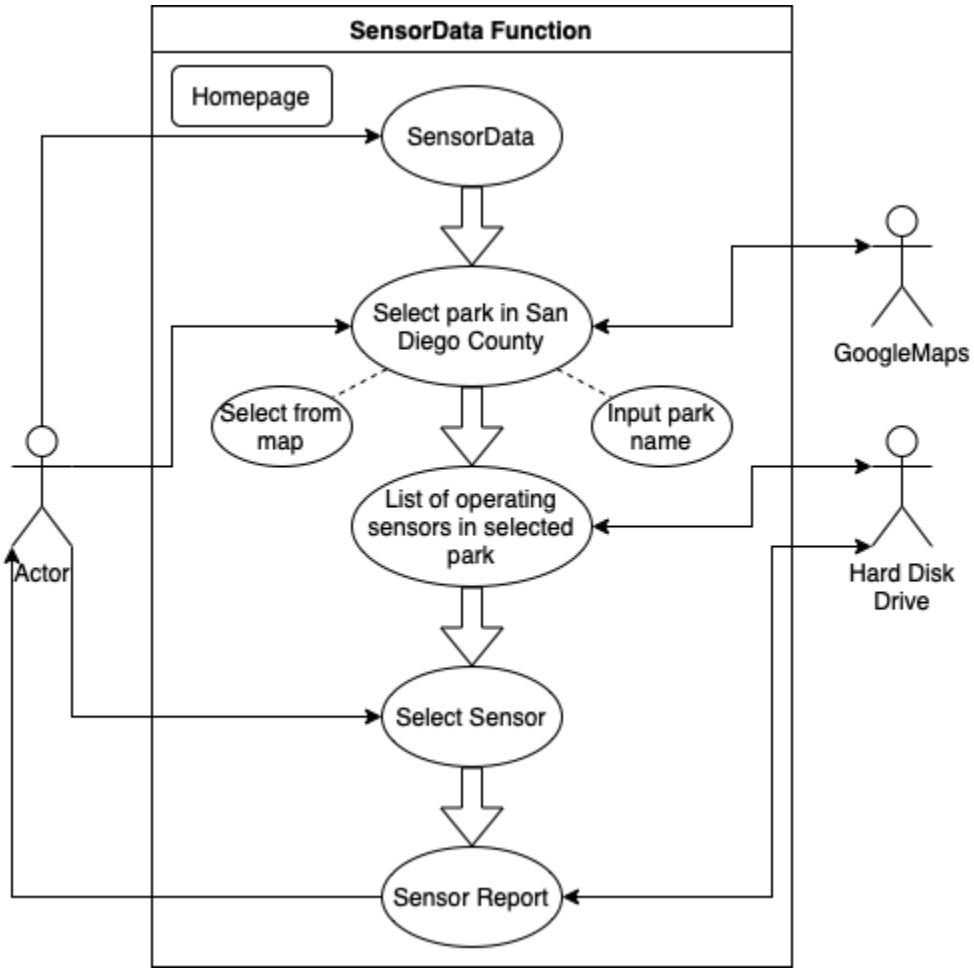
- With the “Catalog” function, the user is able to request a report on all mountain lion detections, viewing or modifying data as needed. Every mountain lion detection that has been picked up by the noise sensors, along with its date and level of classification, is stored in the computer’s hard disk drive. When this function is selected, the user will be redirected to a window displaying a report where all the mountain lion detections for that day and the last 30 days are stored as text files. The user can select the ordering of the data, which can be by date or by level of classification. There are 3 levels to this classification: false, suspected, and definite. “False” (1) is when the ranger has confirmed that the noise sensor inaccurately detected a mountain lion. “Suspected” (2) is when there is no feasible way for the ranger to confirm whether or not the sensor accurately detected a mountain lion. “Definite” (3) is when there is full confirmation by the ranger that a mountain lion was accurately detected. An edit option will allow the user to modify any data they may see as incorrect, and allow them to add new data if need be. To view data in the system older than 30 days, the user will have to select the archive feature to open up a different window where mountain lion detections up to the last 12 months are stored.



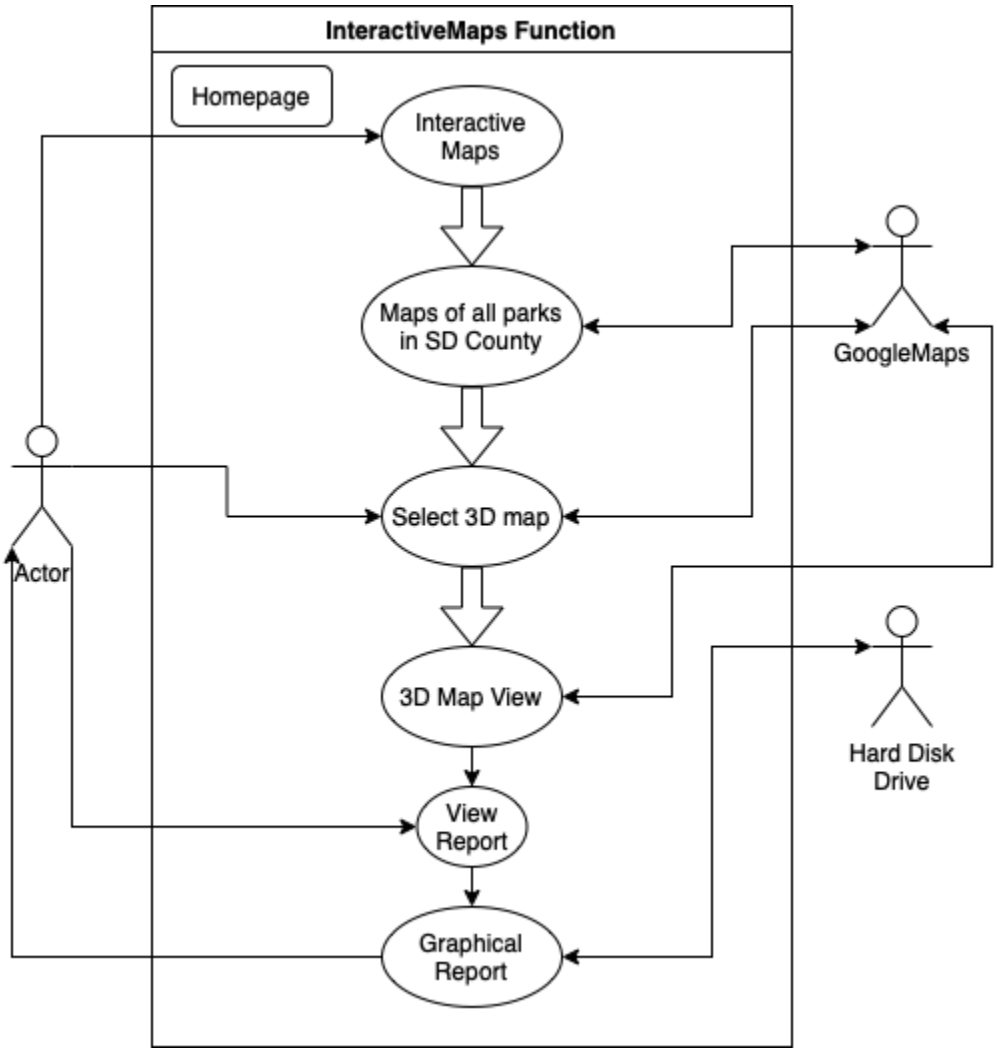
- The client informed us that a park ranger must always open up a new file case for any alerts received by the noise detection sensors. We designed our system so that anytime a new alert message is received by the controlling computer, a new file is automatically opened for the user to fill in with information about the alert. Before filling in the file with their analysis, the park rangers must turn off the alarm manually with an off-switch. A new file will then appear on screen and the user can begin their analysis. The new file opened up by the computer is automatically filled with the day's date, the time of detection, and the name of the park from which the noise was received. After the park ranger has finished their analysis, they will identify the level under which that noise falls in one of the 3 levels of classification: false, suspected, or definite. Once the level of classification has been assigned, the user will save the file's contents onto the hard disk drive, which will then automatically add the file to the catalog. Each new noise detection is reported in its own separate file, for one file cannot have two detections.



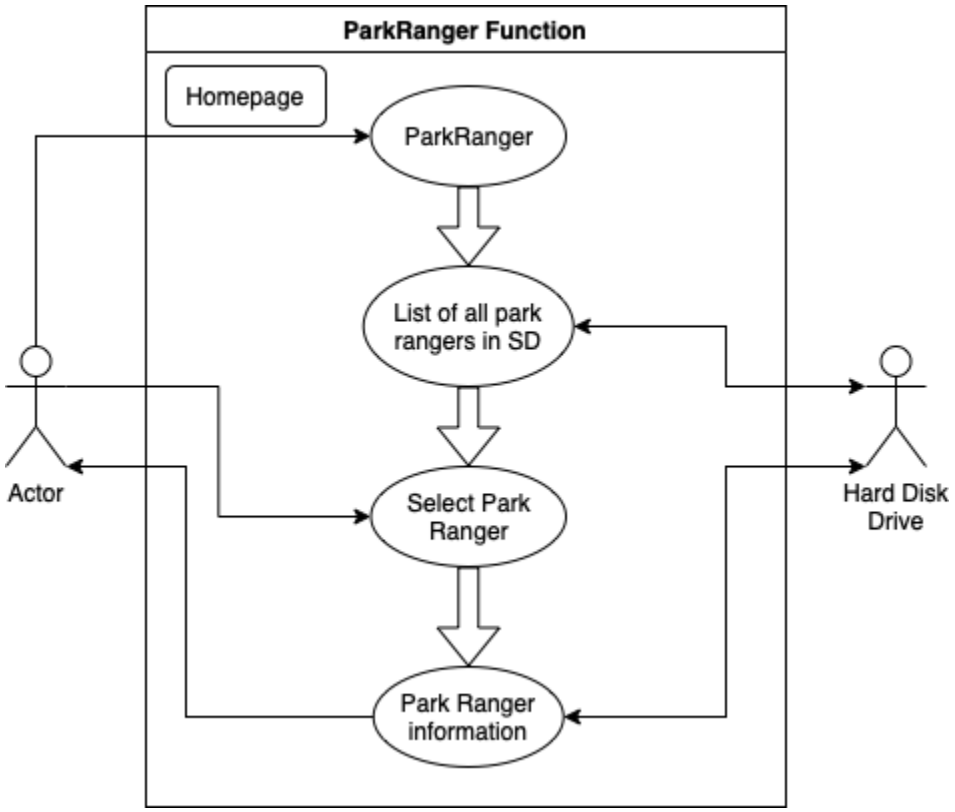
- There will be a “SensorData” function as well. When the user selects this function, a map of San Diego County will appear on screen, from which the user must select a park on the map or input its name manually if not found. After the user has chosen the park or typed in its name, a pop-up window will appear on screen with a list of every sensor currently functional and operating in the selected park. From here, the user will have to select an operating sensor from the list to view information about it. After they have selected the sensor to view, the program outputs a report that pops up on-screen, this report contains all the mountain lion detections for the area monitored by that specific sensor. All relevant information is pulled from the hard disk drive.



- In addition, there will be an “InteractiveMaps" function. The purpose of this function is to allow the user of the program to be able to view 3D-accurate maps of any park or recreation center they choose, and visually pinpoint on these maps where the most frequent mountain lion detections have occurred. When this function is selected, a list containing maps of all of the parks and recreation centers in San Diego County will pop up on screen. Once the user has selected the specific map they want to view, a window will pop up with a 3D view of the selected map. Near the top of the screen there will be an option that reads ‘view report’. After this option has been selected, the hard disk drive will gather all of the relevant information and compile it into a graphical report that will be outputted on the user’s screen. This report will contain a copy of the selected map with visual pins confirming the exact and specific locations where mountain lions were detected within the last 6 months by sensors. In addition, The report will also show any detections from areas within two miles of the selected map’s borders.



- Another function available to the user is the ability to request a report on the classification levels assigned by park rangers during their analysis of noise detection. On the homepage there will be a “ParkRanger” function, once the user selects this function they will be redirected to another window where the names of every active park ranger in San Diego County will be listed, with the information being pulled from the computer’s hard disk drive. Each park ranger will have a file linked to their name. Clicking the park ranger’s name will activate the link and output a report on screen showing the number of mountain lion detections that specific park ranger has analyzed, the level of classification assigned to each detection, and the dates that those detections took place.



Section 2.1b - Non-Functional Requirements

For non-functional requirements, we have been tasked by our client to make sure that the software system we are designing is secure and reliable for the user. Therefore, we designed a login system that aims to provide privacy and security for our client. The login system will require a username that is unique to every staff member in order for them to gain access, as well as their own personal password. Staff will also be able to use their work emails to login as well if they choose to. For reliability, the computer will be connected by wires to a new router designed for a greater network connection for the system. This router will allow for faster internet speeds and quicker reaction time which will be greatly beneficial for the user. In addition, there will also be a database to which the computer will be connected; the purpose of this database is to store all information that passes through the controlling computer. Before being saved onto the database, the information will be saved and stored on the computer's hard disk drive. The hard disk drive will allow the user of the computer to request data and information from the computer directly and quickly rather than having to go through the database and extract it. The software system we are designing is based on the animal detection system developed by the Animals-R-Here company. The detection system includes a number of noise sensors all set up within an area of 5 square miles; the client has specified that there will be no motion sensors or cameras involved, as they are solely focusing on noise detection. The number of noise detection sensors was not specified by the client, however they encouraged us to use 25-35 sensors to cover areas of the park measuring 5 square miles. These noise detection sensors will be programmed to detect various types of animal noises that are native to San Diego County. The purpose of this feature is for the sensor to be able to differentiate between the different types of sounds that it may encounter, and not just perceive a random noise as that of a mountain lion.

Section 2.2 - Validation of System

As specified by our client, our system's overall goal is to generate 4 types of reports: a report showing all mountain lion detections by date detected and by classification (definite, suspected, or false), a report showing all mountain 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, and a report showing detection classifications by ranger.

Verification Test #1: Catalog Report

- User turns computer on and is prompted for login info, then enters their credentials
    - 'Please Login To Continue'
        - Username: "DZuno1991@gmail.com"
        - Password: "PandaBlock223"
    - [Access Granted] … *logging in*
- User is directed to the homepage of the system
    - 'Welcome to the Control Computer, please select an option'
        - Catalog
        - Sensor Data
        - Interactive Map
        - Park Rangers
- User then selects the "Catalog" option
    - *[select]* … 'Catalog'
- User redirected to window displaying report of detections from last 30 days
    - *Mountain Lion Detection Report*
    - 'Sort By:_____ (enter Date, enter Classification Level)'
        - | Archive |
        - MLDECSYS_1_12.08.22.txt {Format: File Name…Class Level…Date}
        - MLDECSYS_3_12.07.22.txt
        - MLDECSYS_3_12.07.22.txt
        - MLDECSYS_2_12.05.22.txt
        - …etc.
- If selected, Archive redirects user to window where data (30 days - 12 months) is stored
    - *Archived Mountain Lion Detection Report*
    - 'Sort By:_____ (enter Date, enter Classification Level)'
        - MLDECSYS_3_11.08.22.txt {Format: File Name…Class Level…Date}
        - MLDECSYS_1_11.03.22.txt
        - MLDECSYS_2_11.01.22.txt
        - MLDECSYS_3_10.29.22.txt
        - MLDECSYS_2_10.29.22.txt
        - …etc.

<<<This Test is a success, we were able to generate a Catalog Report>>>

Verification Test #2: Sensor Report

- User turns computer on and is prompted for login info, then enters their credentials
    - ‘Please Login To Continue’
        - Username: “DZuno1991@gmail.com”
        - Password: “PandaBlock223”
    - ‘Access Granted’ … *[logging in]*
- User is directed to the homepage of the system
    - ‘Welcome to the Control Computer, please select an option’
        - Catalog
        - Sensor Data
        - Interactive Map
        - Park Rangers
- User selects the “Sensor Data” option
    - *[select]* … ‘Sensor Data’
- User is shown 2D-accurate Map of San Diego County
    - ‘San Diego County Map’
    - ‘Please Select Park/Recreation Center or Search Using Name’
        - ‘Search Bar:_____ … (enter Park, enter Rec Center)’
        - ~2D Map of San Diego County~
- Pop-up window appears with list of every operating sensor in selected park
    - ‘Currently Operating Sensors’
        - Sensor#096
        - Sensor#097
        - Sensor#098
        - Sensor#099
        - Sensor#100
        - …etc.
- User selects sensor from list
    - *[select]* … ‘Sensor#099’
- User is shown Sensor Data Report
    - *Sensor Data Report*
        - ‘Mountain Lion Detections picked up by Sensor’
            - SensorDetection#00001
            - SensorDetection#00002
            - SensorDetection#00003
            - SensorDetection#00004
            - SensorDetection#00005
            - …etc.

<<<This Test is a success, we were able to generate a Sensor Data Report>>>

Verification Test #3: Graphical Report

- User turns computer on and is prompted for login info, then enters their credentials
    - ‘Please Login To Continue’
        - Username: “DZuno1991@gmail.com”
        - Password: “PandaBlock223”
    - [Access Granted] … *logging in*
- User is directed to the homepage of the system
    - ‘Welcome to the Control Computer, please select an option’
        - Catalog
        - Sensor Data
        - Interactive Map
        - Park Rangers
- User selects the “Interactive Map” option
    - *[select]* … ‘Interactive Map’
- User is shown list of all Parks and Recreation Centers in San Diego County
    - ‘Regional Parks and Recreation Centers in San Diego County’
    - ‘Please Select Option or Search Using Name’
        - ‘Search Bar:_____ … (enter Name)’
        - Otay Valley Regional Park
        - Mission Trails Regional Park
        - Mira Mesa Recreation Center
        - San Carlos Recreation Center
        - …etc.
- User selects option from list
    - *[select]* … ‘Mission Trails Regional Park - San Diego, CA’
- Pop-up window appears with 3D view of selected park/recreation center
    - ‘Mission Trails Regional Park - San Diego, CA’
        - | View Report |
        - ~3D-Map View of Park/Rec Center~
- User selects “View Report” option near top of screen
    - *[select]* … ‘View Report’
- User is shown Graphical Report
    - *Graphical Report*
        - Detections within Park/RecCenter
            - MLDetection#00001
            - …etc.
        - Detections within 2 miles of Park/Rec Center
            - MLDetection#00525
            - …etc.

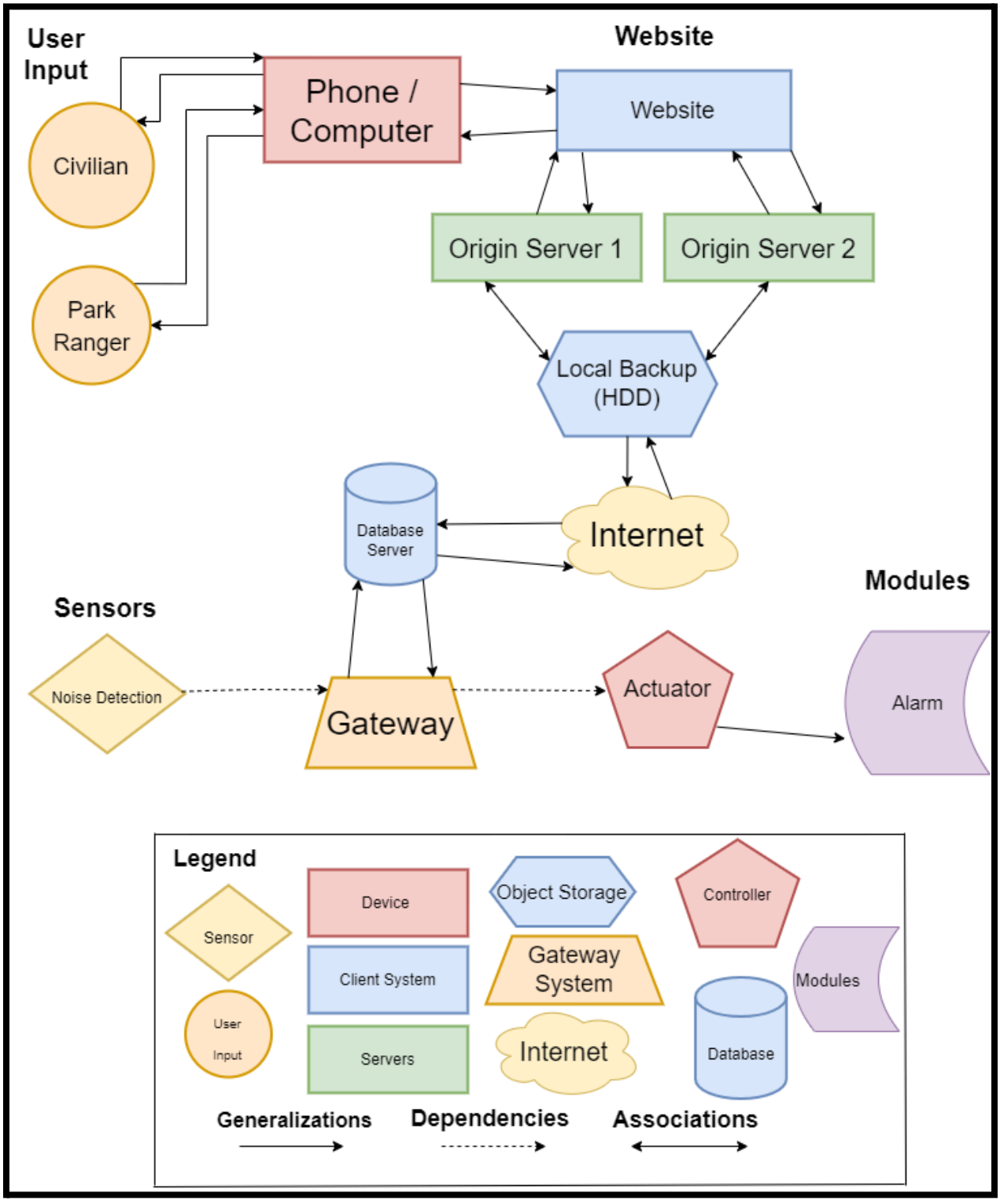
<<<This Test is a success, we were able to generate a Graphical Report>>>

Verification Test #4: Park Ranger Report

- User turns computer on and is prompted for login info, then enters their credentials
    - ‘Please Login To Continue’
        - Username: “DZuno1991@gmail.com”
        - Password: “PandaBlock223”
    - [Access Granted] … *logging in*
- User is directed to the homepage of the system
    - ‘Welcome to the Control Computer, please select an option’
        - Catalog
        - Sensor Data
        - Interactive Map
        - Park Rangers
- User selects the “Park Ranger” option
    - *[select]* … ‘Park Ranger’
- User is shown list of all currently active Park Rangers in San Diego County
    - ‘Currently Active Park Rangers in San Diego County’
    - ‘Please Select Ranger or Search Using Name/ID Number’
        - ‘Search Bar:_____ … (enter Name, enter ID)’
        - Alejandro Botero
        - Jovany Carvajal
        - Luis Orozco
        - Sebastian Sanchez
        - Daniel Zuno
        - …etc.
- User selects Park Ranger from list
    - *[select]* … ‘Daniel Zuno’
- User is shown Park Ranger Report
    - *Park Ranger Report*
        - ‘Name: Daniel Zuno’
        - ‘Ranger ID: 5550014449’
        - ‘Email: DZuno1991@gmail.com’
        - ‘Phone Number: 619-477-2008’
        - ‘Mountain Lion Classifications’
            - Classification#0001
            - Classification#0002
            - Classification#0003
            - Classification#0004
            - Classification#0005
            - …etc.

<<<This Test is a success, we were able to generate a Park Ranger Report>>>

Section 2.3 - Architecture Diagram



Section 2.3a - User Input of the System

- *Civilians Access:*
  The user will have access to the system via cellular device or computer. They have the ability to input data into the website. The user is also able to receive data that is on the website.
- *Park Ranger Access:*
  Park Rangers have special privileges on the website. They will have their user data implemented into the website. These park rangers will be able to access and edit certain HTML files that the normal civilian cannot.
- *Website Access:*
  The website will display information so that the user is on track of the events for mountain lions in the area. Interactive map for the user so access through the website. The Interactive map will be incorporated with Javascript in order to allow the user responsive interaction. CSS will style the website.

Section 2.3b - Website System and Servers

- *Website System:*
  The website system incorporates all the data from the server into the website. The data obtained from the website gets transmitted to the servers. The system is responsive and user friendly.
- *Server System:*
  The server system processes information and responds to incoming internet requests. This is responsible for serving content for an internet property on the website. Latency is not a primary concern here. The data in the server is stored to local storage (HDD).

Section 2.3c - Transmitting to the Database via Internet

- *Local Storage (HDD):*
  The local storage exists as a way to store all the data that has been sent to the servers. Servers 1 & 2 will have the same local storage to keep data organized. The storage system will send data to the main database server via the Internet. The data held will be cleared after a certain amount of time on the local storage.
- *Main Database Server:*
  This main database server stores information transmitted from the local storage (HDD). The main database server will maintain all the precious data for the organization. It is managed and only authorized by trusted users. This database has all the digital files and information. The main database receives data from the gateway which contains sensor data. The database will include functions into the gateway. This is how the danger alerts will be uploaded to the website. The sensor data from the database will be uploaded to the local storage (HDD) which is accessed by the servers.

Section 2.3d - Sensor Detection Gateway

- *Sensors:*
  There will be noise sensors in order to detect mountain lions. The noise sensors will detect based on animal noises. These sensors will send an alert message that goes towards the gateway.
- *Gateway:*
  The gateway exists as a way to transmit the sensor data. The sensor data will now be sent to the database. The database also incorporates functionality into the gateway. The gateway also sends the data from the sensors to the actuators.

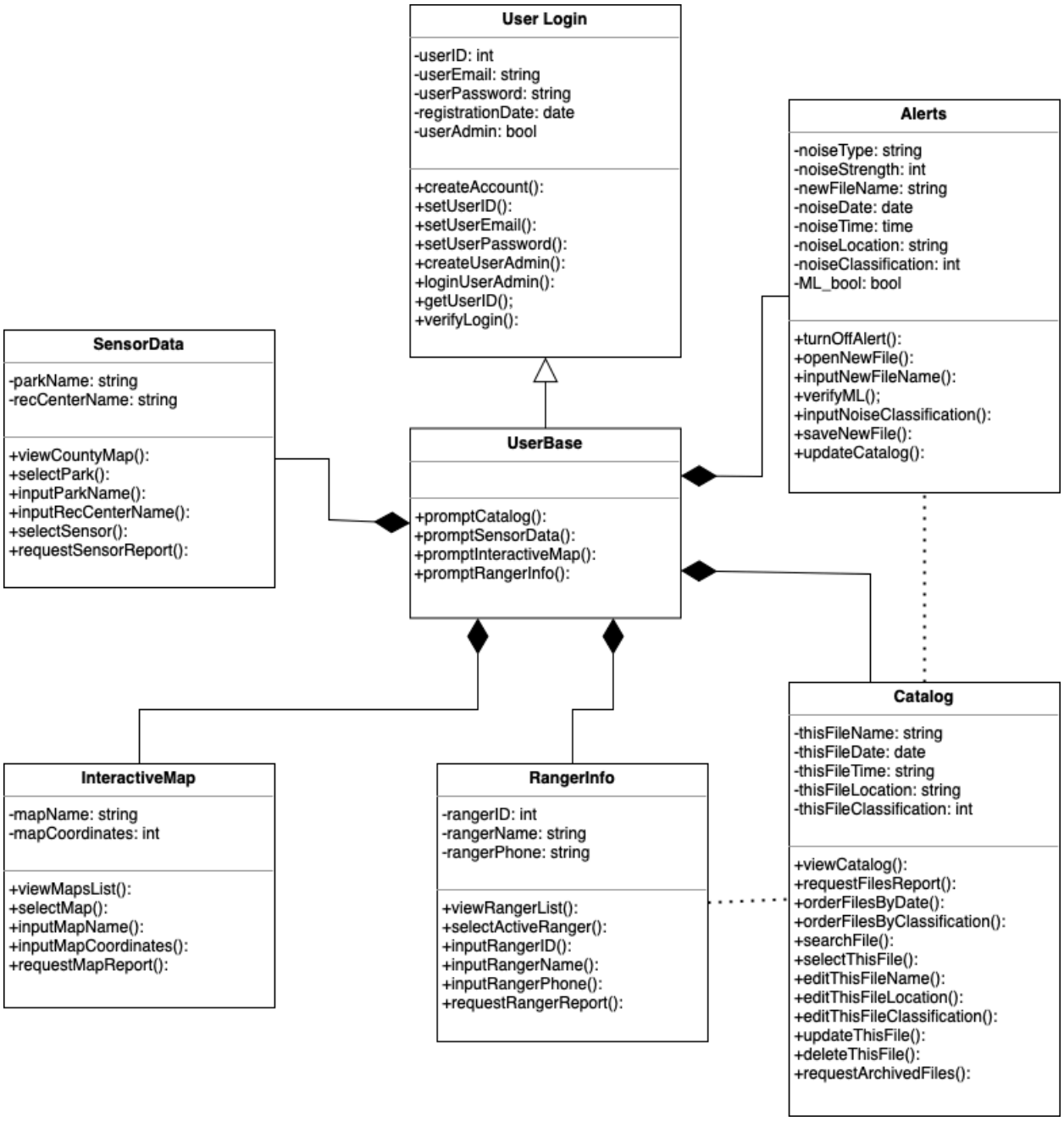
Section 2.3e - Alarm System Access:

- *Actuator:*
  Actuators exist in order to trigger the alarm system. The actuators require the sensor data in order to activate. It will receive that sensor data from the gateway. The actuator contains a computerized unit that tells it how to function. The computerized unit will be processing the sensor data.
- *Alarm System:*
  The alarm system will cause the alarm to turn off based on the input from the actuators. The alarm system will have a manual off-switch. The actuator will activate the device causing the alarm to activate.

Section 2.4 - System Security

Since the day the first computer was created, hackers have been looking for new ways to exploit, change, steal, or delete information stored on these systems. The motivation for these hackers may be related to reasons such as financial gain, vandalism, politics, and espionage to name a few, but there exist many more. As a result we have implemented a variety of security measures to protect our users from these malicious acts. First off, we have installed a firewall that will prevent unauthorized access to the system's network and alert the user about any intrusion attempts by outsiders; this firewall was purchased from CISCO Systems Inc. and has been modified to fit our system. In addition to the firewall, we have installed an antivirus software known as ESET NOD32 to protect against unauthorized code or software that poses a serious threat to our operating system's capabilities. Thankfully, viruses may have easy-to-spot effects such as slowing down the computer or deleting key files. Our antivirus software will play a major role in protecting our system by detecting real-time threats and providing automatic updates to new threats in order to ensure that the system's data remains safe. Finally, we installed an encryption software known as Veracrypt in order to encrypt user information stored on the system. Hackers are constantly looking for new ways to exploit systems every day, so having encryption software to protect sensitive information is extremely important because firewalls and antiviruses are not invincible, they can still be exploited and bypassed by professional hackers.

Section 2.5 - Class Diagram



Section 2.5a - Description of Classes, Attributes, and Operations

- ***User Login Class:*** The User Login Class is what makes up the lock screen of the system. Here, the user will be prompted to enter their ID number and password for verification in order to login and access the computer's contents. Every staff member will have their own personal username for privacy reasons, and they can also use their work email to login as well. Once the user enters their username or email on the lock screen, it will be run through the system authentication service to verify credentials. Once this process is finished and the system gives the user clearance, the user will be prompted to enter their

password. Once the staff member's username and password have both been cleared by the system, the user will be granted access to the computer's contents.

- **Attributes:** There are 5 attributes for this class. The first attribute is 'userID', representing the user's login ID number; this attribute is an integer data type. The second attribute is 'userEmail', representing the user's login work email that they can use instead of their ID; this attribute is a string data type. The third attribute is 'userPassword', representing the user's login password that they use to sign in; this attribute is a string data type. The fourth attribute is 'registrationDate', representing the date that the user created their account. The fifth attribute 'userAdmin' returns true only if the user logins to their admin portal using a special code.
- **Operations:** There are 8 operations for this class. The 'createAccount()' account operation allows the user to create a new account. The 'setUserID()' operation allows the user to set up their new ID after selecting the option to create an account. The 'setUserEmail()' operation allows the user to set up an email for their account. The 'setUserPassword()' operation allows the user to set up their confidential password. The 'createUserAdmin()' operation allows one administrator to set up a new administrator account. The 'loginUserAdmin()' operation allows the user to login to their admin portal. The 'getUserID()' operation returns the user's ID number by sending a unique access code to their email address. The 'verifyLogin()' operation verifies the user's credentials and password by running them through the system authentication service. Once the credentials and password have been verified, the user will be granted access.

- ***UserBase Class:*** The UserBase Class is the main homepage for the user. Here, the user will be able to request several different reports, including: a report showing all mountain lion detections by date detected or by classification (false, suspected, definite), a report showing all mountain 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, and a report showing classifications by park ranger.
  - **Attributes:** This class has no attributes, only operations
  - **Operations:** There are 4 operations for this class. The 'promptCatalog()' operation allows the user to access the computer's catalog. The 'promptSensorData()' operation allows the user to access the data of specific noise sensors. The 'promptInteractiveMap()' operation allows the user to view maps with locator pins, each pin representing mountain lion detections in the area. The 'promptRangerInfo()' operation allows the user to access the data of specific Park Rangers who are currently active.
- ***Catalog Class:*** The Catalog Class is where the user is able to view and modify information that has been stored in the database. Every mountain lion detection that has been picked up by the noise sensors is automatically stored on the computer's hard disk

drive along with its date, time, location, and level of classification. The main purpose here is to allow the user to request a report on all mountain lion detections ordered by date or level of classification. When this function is selected, the user will be redirected to a window where all mountain lion detections for that day and the last 30 days are stored as files. The files will be inaccessible until they have requested a findings report, after which they must select the order in which the files are shown, either by date or by level of classification. There are 3 levels: false, suspected, and definite. “False” means a ranger has confirmed that the noise sensor inaccurately detected a mountain lion. “Suspected” means there is no feasible way for the ranger to confirm whether or not the sensor accurately detected a mountain lion. “Definite” means there is full confirmation by the ranger that a mountain lion was accurately detected. After the order is chosen, the user will be able to select any file and open it to view or edit contents. If the user edits any file, they must update the file before closing it. Once the file is closed and the user is done, the catalog will automatically be updated and the program may be exited. To view or modify data in the system older than 30 days, the user will select the archive operation to open up a separate window where mountain lion detections up to the last 12 months are stored.

- **Attributes:** There are 5 attributes for this class. The first attribute is ‘thisFileName’, representing the selected file’s name; this attribute is a string data type. The second attribute is ‘thisFileDate’, representing the selected file’s date that it was created by a park ranger. The third attribute is ‘thisFileTime’, representing the time of day the selected file’s noise was detected by a sensor; this attribute is a string data type. The fourth attribute is ‘thisFileLocation’, representing the location (park/recreation center) where the selected file’s noise was detected; this attribute is a string data type. The fifth attribute is ‘thisFileClassification’, representing the selected file’s level of classification previously assigned by another ranger; this attribute is an integer data type.
- **Operations:** There are 12 operations for this class. The ‘viewCatalog()’ operation allows the user to view a list of files of all the mountain lion detections stored on the computer within the last 30 days. The ‘requestFilesReport()’ operation allows the user to request a report showing all mountain lion detections by date detected or by classification. The ‘orderFilesByDate()’ operation allows the user to change the ordering of the files in terms of date detected. The ‘orderFilesByClassification()’ operation allows the user to change the ordering of the files in terms of classification level. The ‘searchFile()’ operation allows the user to manually search a file using the search bar. The ‘selectThisFile()’ operation allows the user to select a specific file that they want to open or edit. The ‘editThisFileName()’ operation allows the user to edit the name of their selected file. The ‘editThisFileLocation()’ operation allows the user to edit the location (park/recreation center) of the selected file where a mountain lion noise

was detected. The ‘editThisFileClassification()’ operation allows the user to edit the classification level for the selected file previously assigned by another ranger. The ‘updateThisFile()’ operation allows the user to save the updates/changes made to the selected file. The ‘deleteThisFile()’ operation allows the user to delete the selected file. The ‘requestArchivedFiles()’ operation allows the user to request that the computer output files older than 30 days, but younger than 12 months.

- ***Alerts Class:*** The system is designed so that any new noise detected by the noise sensors sends an alarm signal with a message to the controlling computer located in the park ranger station. This alarm message contains the type of noise detected, the strength of the noise, along with the time and location where it was detected. Before anything else, the park ranger must turn off the sounding alarm manually via an off-switch. Once that has been done, a message will appear on screen prompting them to open a new file. The new file that is opened by the park ranger is automatically filled by the computer with the type of noise detected, the strength of the noise, and the time it was detected. This information helps the park ranger to complete their analysis, where they must assign a level of classification to the noise. After analysis, the file must then be manually saved and then added to the catalog by the park ranger, which will also store it in the database.
    - **Attributes:** There are 8 attributes for this class. The first attribute is ‘noiseType’, representing the type of new noise detected by the sensor; this attribute is a string data type. The second attribute is ‘noiseStrength’, representing the strength of the new noise detected by the sensor; this attribute is an integer data type. The third attribute is ‘newFileName’, representing the name given to a new file created by a ranger for the new noise; this attribute is a string data type. The fourth attribute is ‘noiseDate’, representing the date that the new file was created by a park ranger. The fifth attribute is ‘noiseTime’, representing the time of day that the new noise was detected by a sensor; this attribute is a string data type. The sixth attribute is ‘noiseLocation’, representing the park or recreation center where the new noise was detected by the sensor; this attribute is a string data type. The seventh attribute is ‘noiseClassification’, representing the level of classification assigned to the new noise by a park ranger; this attribute is an integer data type. The eighth attribute is ‘ML_bool’, representing whether or not it is true or false that a mountain lion was detected by the noise sensors.
    - **Operations:** There are 7 operations for this class. The ‘turnOffAlert()’ operation allows the user to manually turn off the alert signal that goes off anytime a new noise is detected. The ‘openNewFile()’ operation allows the user to open a new file once an alert message has been received. The ‘inputNewFileName()’ operation allows the user to assign the new file a name. The ‘verifyML()’ operation allows the user to run the noise verification system that assists them in analyzing whether or not the noise detected by a sensor came from a mountain

lion. The ‘inputNoiseLocation()’ operation allows the user to assign the new file a location (park/recreation center) where the newly detected noise came from. The ‘inputNoiseClassification()’ operation allows the user to assign the new file a level of classification. The ‘saveNewFile()’ operation allows the user to save the new file and its contents. The ‘updateCatalog()’ operation allows the user to add the new file to the computer’s catalog, which stores it in the database as well.

- ***SensorData Class:*** The SensorData class will allow the user to request a report on all the noises detected by specific sensors in parks and recreation centers around the county. When the user selects this function, a map of San Diego County will appear on screen, and from this map the user must select a park in the area. If they are looking for a specific park or recreation center, then they will be able to input its name manually in the search bar. After the user has selected the park, or recreation center, from the map or typed in its name manually in the search bar, a window will pop-up on screen with a list of every sensor currently functioning and operating in that selected location. From here, the user will have to select an operating sensor from the list to view information about it. After a sensor has been selected, the program will prompt the user with an option, “Request Report”. The “Request Report” option, when selected, outputs a findings report that contains all of the noise detections for the area that specific sensor monitors. All relevant information is pulled from the hard disk drive.
    - **Attributes:** There are 2 attributes for this class. The first attribute is ‘parkName’, representing the name of the park from which the user will select a noise sensor to view; this attribute is a string data type. The second attribute is ‘recCenterName’, representing the name of the recreation center from which the user will select a noise sensor to view; this attribute is a string data type.
    - **Operations:** There are 6 operations in this class. The ‘viewCountyMap()’ operation allows the user to view a map of San Diego County where they can select a location (park/recreation center) of their choosing. The ‘selectPark()’ operation allows the user to manually select a location on the map provided. The ‘inputParkName()’ operation allows the user to search the name of a park using the search bar if they cannot find it on the map. The ‘inputRecCenterName()’ operation allows the user to search the name of a recreation center using the search bar if they cannot find it on the map. The ‘selectSensor()’ operation allows the user to select a sensor from a location (park/recreation center) that they have chosen on the map. The ‘requestSensorReport()’ operation allows the user to request a report showing all mountain lion detections at a specific sensor location.
- ***InteractiveMap Class:*** The purpose of the InteractiveMap Class is to allow the user to be able to view accurate maps of any park or recreation center in San Diego County, and visually pinpoint on these maps where the most frequent mountain lion detections have occurred. When this function is selected, a list containing maps of all the parks and recreation centers in San Diego County will appear on screen, with the user having to

select a map that they want to view from this list. In addition to simply going through the list and selecting an option, the user may also manually type in the name or coordinates of the park/recreation center using the search bar. Once the user has selected the specific map they want to view, a 2D-view of the selected map will pop-up on screen via GoogleMaps, with the user being able to toggle around with the features as if they were directly using the app, even being able to use features such as 3D mode. Near the top-right corner of the map's screen will be an option that reads "Request Map Report". When this option has been selected, a copy of the selected map will be outputted on screen with several pins located all over the map, and a report containing all the relevant information about each pin. These visual pins represent the exact and specific locations at this park/recreation center where mountain lions were detected by noise sensors within the last 6 months. In addition, the map and report will show any detections from areas within two miles of the selected map's borders.

- **Attributes:** There are 2 attributes for this class. The first attribute is 'mapName', representing the name of the park or recreation center that the user wants to view a map of; this attribute is a string data type. The second attribute is 'mapCoordinates', representing the coordinates of the park or recreation center that the user wants to view a map of; this attribute is an integer data type.
- **Operations:** There are 5 operations for this class. The 'viewMapsList()' operation allows the user to view a list of maps with locator pins detailing mountain lion detections. The 'selectMap()' operation allows the user to select a map from the list that they want to view. The 'inputMapName()' operation allows the user to search for a mapped location using the search bar if they cannot find it in the list. The 'inputMapCoordinates()' operation allows the user to search for a mapped location using the search bar if they cannot find it in the list. The 'requestMapReport()' operation allows the user to request a graphical report showing detections on a map of the location and areas within 2 miles of the park.

- ***RangerInfo Class:*** The RangerInfo Class allows the user to request a report on the classification levels assigned by park rangers during their analysis of noise detections. When this function is selected, a window will pop-up on screen with the list of names of every active park ranger in San Diego County. Clicking on the name of a park ranger prompts the program to output a file showing relevant work information such as the park ranger's rank, work schedule, number of years worked, current duties, etc. Each park ranger will also have an option called "Request Report" linked next to their name. Selecting this option will output a report on screen showing the number of noise detections that specific park ranger has analyzed, the level of classification assigned to each detection, and the dates that those detections took place.
  - **Attributes:** There are 3 attributes for this class. The first attribute is 'rangerID', representing the ID number of the park ranger that the user wants to look up; this attribute is an integer data type. The second attribute is 'rangerName',

representing the name of the park ranger that the user wants to look up; this attribute is a string data type. The third attribute is ‘rangerPhone’, representing the phone number that is used to call the park ranger in case of an emergency situation.

- **Operations:** There are 6 operations for this class. The ‘viewRangerList()’ operation allows the user to view the list of all currently active park rangers in San Diego County. The ‘selectActiveRanger()’ operation allows the user to select one ranger from the list and view their information. The ‘inputRangerID()’ operation allows the user to search for a specific park ranger with their ID number using the search bar if they cannot find them in the list. The ‘inputRangerName()’ operation allows the user to search for a specific park ranger with their full legal name using the search bar if they cannot find them in the list. The ‘inputRangerPhone()’ operation allows the user to to search for a specific par ranger using their phone number. The ‘requestRangerReport()’ operation allows the user to request a report showing classifications by park ranger.

Section 2.6 - Verification Method

Section 2.6a - Unit Tests

Unit Test #1: *<< inputRangerID(int) : bool >>*

- Allows user to sort list and select specific park ranger using their ID number
- Only allows to select/sort one park ranger at a time
- ID number must be at least 10 numbers in length
- Gives user the option to view report for park ranger selected
- Returns (bool = True) when finished

- **[Test Case 1 for Unit Test #1]**
  - InputRangerID(5550014449); //user enters ID number for park ranger
  - Method sorts through park ranger list to select matching ID number
  - Program Output: "Ranger: Zuno, Daniel ~ ID: 5550014449 ~ Ranger Report"
  - ViewInfo allows user to view park ranger's work on mountain lion detections
  - Returns (bool = True)
  - From these results, we conclude that our unit test was successful
- **[Test Case 2 for Unit Test #1]**
  - InputRangerID(35422217); //user enters wrong ID number for park ranger
  - Program Output: "Error: Invalid Syntax - incorrect number of characters"
  - Method does not sort through park ranger because no ID exists with 8 numbers
  - ViewInfo does not appear on screen as a result of invalid syntax
  - Returns (bool = False)
  - From these results, we conclude that our unit test was successful

Unit Test #2: *<< setUserEmail(string): null → getUserEmail(): string >>*

- Allows user to set and get their login Email for main computer
- Email must have all valid characters (no spaces or symbols)
- Only 1 Email allowed per user, but Email may be changed
- We test this by verifying against a getter function getUserEmail();

- **[Test Case 1 for Unit Test #2]**
  - Call setUserEmail and pass in a string parameter
  - Program Output: "Setup New Email: _____"
  - User types in Email such as: "JohnWayne420@yahoo.com"
  - We test to see if our modifier worked on our user object by performing a unit test
  - We call getUserEmail() in our test suite, expecting a string with values
  - Program Output: "Your login Email is: JohnWayne420@yahoo.com"
  - Emails are the same, therefore we know our setter and getter functions work
  - From these results, we conclude that our unit test was successful

- **[Test Case 2 for Unit Test #2]**
    - Call setUserEmail and pass in a string parameter
    - Program Output: “Set New Email: _____”
    - User types in Email such as: “John*420.$Wayne#7@yahoo.com”
    - We test to see if our modifier worked on our user object by performing a unit test
    - We call getUserEmail() in our test suite, expecting a string with values
    - Program Output: “Error: Invalid Syntax - using prohibited characters and spaces”
    - Setter and getter functions work but syntax was wrong, rendering email invalid
    - From these results, we conclude that our unit test was successful

Section 2.6b - Function Tests

Function Test #1: *<< verifyLogin(string userEmail, string userPassword): bool >>*

- This function will verify login credentials of user
- User credentials must be verified before user is granted access to main computer
- By passing in two string parameters (userEmail, userPassword), we can set up our test suite to be expecting a boolean value when the user inputs their username and password
- Our test suite contains verifyLogin() and is expecting a boolean value in return. We are going to test 2 cases covering 100% of test output specification.
- System is case-sensitive for both username and password
- Bool value is set to False, will set to true if login credentials match with existing records

- **[Test Case 1 for Function Test #1]**
    - User turns on computer and is prompted for their username/email and password
    - User chooses email option and enters their email, “JohnWayne420@yahoo.com”
    - User enters their password in, “IronMan1998”
    - verifyLogin() function is run, finds that credentials match with existing records
    - Program Output: ‘Access Granted’ … [*logging in]*
    - Returns (bool = True)
    - From these results, we conclude that our function test was successful
- **[Test Case 2 for Function Test #1]**
    - User turns on computer and is prompted for their username/email and password
    - User chooses email option and enters their email, “JohnWayne420@yahoo.com”
    - User spells their password with the wrong casing, “ironman1998”
    - verifyLogin() function is run, but no credentials match with existing records
    - Program Output: ‘Access Denied’ … *[invalid username or password]*
    - Returns (bool = False)
    - From these results, we conclude that our function test was successful

Function Test #2: *<< verifyML(string noiseType, int noiseStrength): bool >>*

- This function assists the park ranger in analyzing a detected noise to confirm the animal
- By passing in two data parameters(noiseType, noiseStrength), we can set up our test suite to be expecting a boolean value when the user runs the verification function
- Multiple noise sensors record animal noises in the parks/rec centers all over SD County
- Once a noise has been recorded, the type of noise and the strength of the noise are determined by the sensor's programming, which then sends the data to the main computer
- Once that information has been sent to the main computer, the park ranger currently present or on duty will run the verification function and will determine if the noise did or did not belong to a mountain lion, and then assign a classification level
- Noise strength is ranged from 1-10, with 1 being lowest decibels, and 10 being highest
- Classification Levels are assigned to noises by park ranger from range 1-3, with 1 being "no, not a mountain lion", 2 being "unsure", and 3 being "yes, it is a mountain lion"
- If the noise is undetermined by the verification function because of system bugs, bad weather, or other external factors beyond our control, the report is cached and reviewed by a ranger that manually overrides the VerifyML() function to conclude the report
- ML_bool is set to False, will set to "True" if noise matches that of a mountain lion

- **[Test Case 1 for Function Test #2]**
  - Sensor picks up animal noise in park or recreation center
  - The type and strength of the noise are sent, along with the date, time, and location
  - On-duty park ranger runs verification function by passing in the parameters
    - VerifyML(growl, 8); //parameters are passed into function
  - Function runs, determines noise belongs to mountain lion
  - Ranger assigns classification level after analysis of function results
    - inputNoiseClassification(int, bool);
    - <Level 3, True>
    - Output: [Detection{5} / October 18, 2022 / 3:29 PM / Torrey Pines Reserve, San Diego / Level 3 / True]
- **[Test Case 2 for Function Test #2]**
  - Sensor picks up animal noise in park or recreation center
  - The type and strength of the noise are sent, along with the date, time, and location
  - On-duty park ranger runs verification function by passing in the parameters
    - VerifyML(hiss, 4); //parameters are passed into function
  - Function runs, determines noise does not belong to mountain lion
  - Ranger assigns classification level after analysis of function results
    - inputNoiseClassification(int, bool);
    - <Level 1, False>
    - Output: [Detection{5} / October 18, 2022 / 3:29 PM / Torrey Pines Reserve, San Diego / Level 1 / False]
  - Looking at both test results, we conclude that our function tests were successful

Section 2.6c - System Tests

System Test #1: *<< updateCatalog(): null >>*

{Before Use}

- Before anything, user must have an administrator role account (or create one) in order to view and declare basic commands in the catalog system
- Anytime a new noise is detected, an alert signal goes off and a message is sent to the main computer with: noise type and noise strength
- After ranger turns off alert signal, computer prompts them if they want to open new file
- If ranger opens new file, they must run the noise verification function → Verify_ML()
- Noise verification function verifies whether or not the detected noise was a mountain lion
- Once noise verification is complete, user is allowed to edit new file and update catalog

{While In Use}

- First, user must already have (or create) an admin role to make basic commands
    - ➢ createUserAdmin(user, pass): bool | loginUserAdmin(user, pass): bool
        - Creates/logs into admin in order to command the system
        - Parameters: username(string), password(string); returns boolean
- Once user has admin, they can go in, view files in catalog, and make changes
- Once main computer receives new alert, noise verification must be called by user
- After noise verification is completed, user adds relevant information to the newly created file and updates the catalog with the new file
    - UpdateCatalog(newFile): null
- This function is accessed by the administrator user only
- Will use ViewCatalog() to test if the catalog has been updated
- Bool value set to False, will set to True once user has logged in or created admin portal

- **[Test Case 1 for System Test #1]**

**[new file] = [unNamed / October 25, 2022 / 6:52PM / Mission Trails, San Diego / null]
loginUserAdmin(John, batman123); //user logs in to their user admin portal

**inputNewFileName(String, bool):
<"Detection{7}", True> //number in parenthesis = nth file that day (ex: 9th file on Oct. 25)
Output: [Detection{7} / True]

**inputNoiseClassification(int, bool):
<Level 3, True>
Output: [Detection{7} / October 25, 2022 / 6:52PM / Mission Trails, San Diego / Level 3 / True]

**saveNewFile(Detection{7});
<Detection{7}, True>
Output: “New File has been saved. Please update the catalog.”

**updateCatalog(Detection{7});
<Catalog, Detection{7}>
Output: [Catalog: …Detection{4}, Detection{5}, Detection{6}, Detection{7}] //catalog updates

- **[Test Case 2 for System Test#1]**

**[new file] = [unNamed / October 25, 2022 / 7:15PM / Otay Valley, San Diego / null]
createUserAdmin(); ~ loginUserAdmin(); //user does not create or login to user admin portal

**inputNewFileName(String, bool):
<”Detection{8}”, False> //number in parenthesis = nth file that day (ex: 9th file on Oct. 25)
Output: “Error: You do not have permission to edit this file, please see administrator.”

**inputNoiseClassification(int, bool):
<3, False>
Output: “Error: You do not have permission to edit this file, please see administrator.”

**saveNewFile(Detection{8});
<Detection{8}, False>
Output: “Error: You do not have permission to save this file, please see administrator.”

**updateCatalog(Detection{8});
<Catalog, null>
Output: [Catalog: …Detection{5}, Detection{6}, Detection{7}] //catalog does not update
From these two test results, we conclude that our system test was successful

System Test #2: *<< requestRangerReport(): null >>*

{Before Use}

- Before anything, user must have an administrator role account (or create one) in order to view and declare basic commands in the park ranger system
- Anytime a ranger completes analysis of a new noise, they save information in a file and save the file to the catalog, which sends the file to the park ranger class
- These files are directly linked to the names of the park rangers in the park ranger class
- Anybody who has a login ID/email can view the list of active rangers in San Diego County, however, only an admin can actually request a report on a specific ranger
- No changes can be made to the park ranger files, files must be changed through catalog

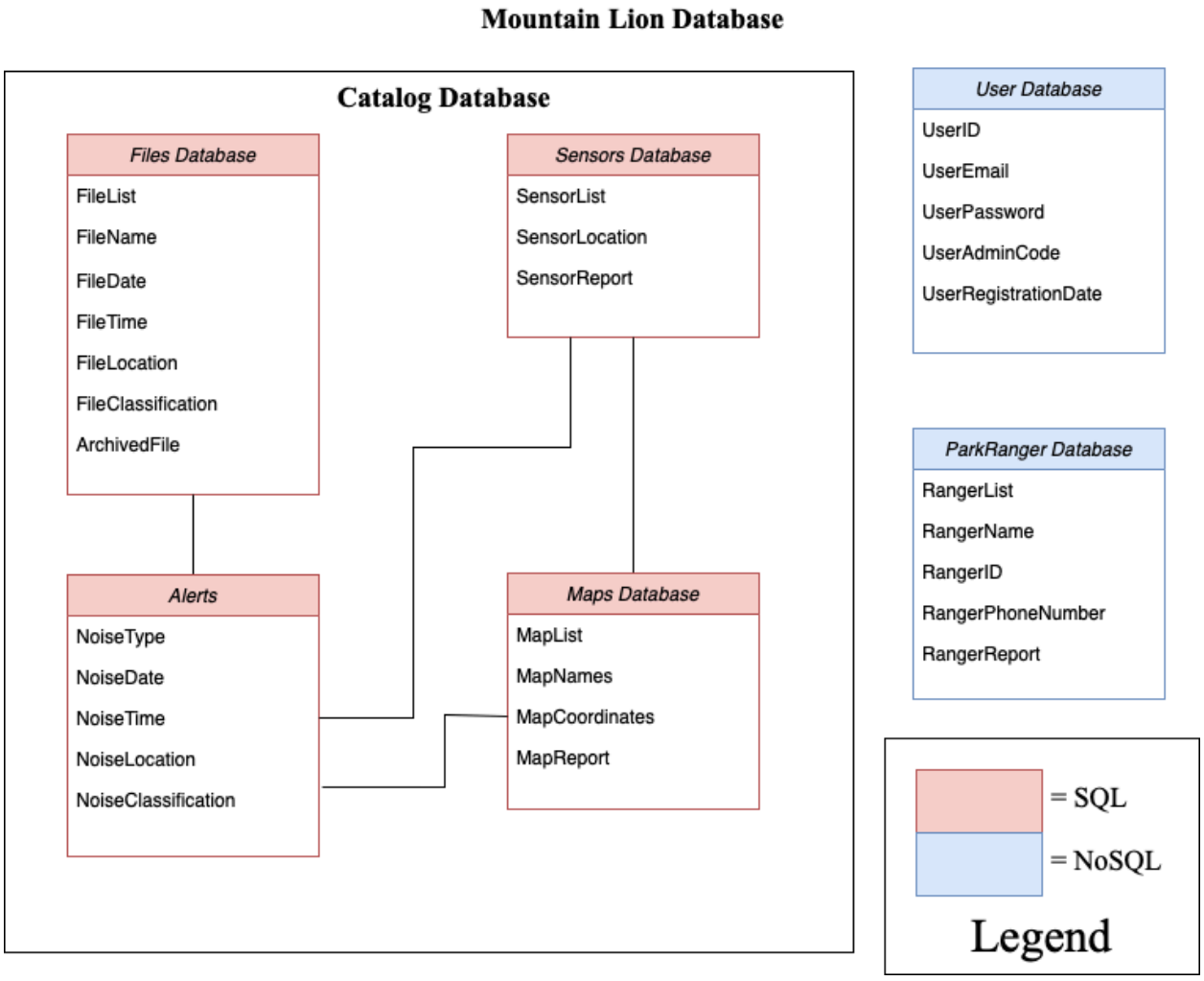
{While In Use}

- First, user must already have (or create) an admin role to make basic commands
    - ➢ createUserAdmin(user, pass): bool | loginUserAdmin(user, pass): bool
        - Creates/logs into admin in order to command the system
        - Parameters: username(string), password(string); returns boolean
- Once user has admin, they can go in and view the list of active park rangers in San Diego
- They can either select a ranger straight from a list, or search them using a name and ID
- Once they have selected a ranger, they will be able to request a report on them
- These reports have a set of information such as: ranger's findings, the analysis files they have completed, and the classification levels they assigned to those files
- To change ranger file information, files must be edited directly from catalog

- **[Test Case 1 for System Test#2]**
    - User logs into main computer using the login system
        - VerifyLogin(JohnWayne420@yahoo.com, IronMan1998)
        - Output: [Access Granted] … "*logging in"*
    - loginUserAdmin(John, batman123); //user logs into their user admin portal
    - User pulls up the ParkRanger class by making a command
        - promptRangerInfo();
    - User is shown list of active park rangers in SD County by making command
        - viewRangerList();
    - User searches for specific ranger using search function
        - inputRangerID(5550014449);
        - Output: "Ranger: Doe, John ~ ID: 5550014449 ~ Ranger Report"
    - User requests a report by clicking on 'Ranger Report'
        - requestRangerReport(); null
        - Output: [Report on 5550014449]
    - Report is shown on screen detailing Ranger's findings, analysis, & classifications
    - From these results, we conclude that our system test was successful

- **[Test Case 2 for System Test#2]**
    - User logs into main computer using the login system
        - VerifyLogin(ClarkKent77@gmail.com, DailyB2013)
        - Output: [Access Granted] … "*logging in"*
    - User does not login to their admin portal
    - User pulls up the ParkRanger class by making a command
        - promptRangerInfo();
    - User is shown list of all park rangers in SD County by making command
        - viewRangerList();
    - User searches for specific ranger using search function
        - inputRangerID(5512108824);
        - Output: "Ranger: Jones, Eric ~ ID: 5512108824 ~ Ranger Report"
    - User requests a report by clicking on 'Ranger Report'
        - requestRangerReport(); null
        - Output: "[Error: you do not have permission to make this command]"
    - Report is not shown on screen
    - From these results, we conclude that our system test was successful

Section 2.7 - Data Management Diagram



Section 2.7a - Database Description

For our Mountain Lion Detection System, the best implementation of a database would be to utilize separate databases that relate with each other. One database will be SQL based and the other database(s) will be NoSQL. We decided to assign it the unique identifying name of ParkDatabase for simplicity. The SQL database will implement all the elements contained in the park such as alerts, sensors, maps, and data files. Each of those will have their own memory spaces for holding data, and will be relationally linked to one another in order to pass and share information. The park rangers are responsible for interpreting any data collected from our sensors located in parks, and using that information to generate their reports.

Alongside our ParkDatabase, we will have another one named UsersDatabase, which will include the records of every user account that has been previously created for our system. Our NoSQL approach will result in a document-style database, which means it will be easier to add new users and park rangers to the system, as well as updating any of their attributes if our system evolves to include more data in the future. For the initial launch, we will simply have our system setup in order to record basic information about our users, mostly for the purpose of logging in.

As for the rangers, their database table will also be approached and implemented using NoSQL. In addition, it will also contain some of their basic user information, as well as the reports that a respective park ranger produces. The park ranger reports are a collection of data compiled by the ranger on mountain lion detections in San Diego County, with information varying depending on their analysis of the situation. Since each report will be different, some more detailed than others, we will be using a dynamically-scaled database language like NoSQL.

Section 2.8 - Future Features

For this software system, the client has requested that we design our system so that it may be portable for future use or reference. Recently, the California State Park system has shown great interest in the detection system developed by the Animals-R-Here company for its efficiency. This interest was great enough that the client has asked for us to build our system in a way that would allow it to be easily reconfigured for a different park in California if needed. In addition to this feature, we have also made our system reconfigurable for different animals. If our system were to be used and tested for another state park in California, it can be reconfigured to detect and store information for animals other than mountain lions. We are assuming the client will not be satisfied with the system immediately and will want updates as time goes on, so we will keep our system flexible in order to meet any new needs and desires of the client. We are also interested in the future addition of a feature which tells the user what time of day that mountain lions are most likely to come out and roam public areas. We are proud of our system and we hope that the first prototype is well-received by the client.

Section 2.9 - Project Timeline

A spreadsheet to visualize the time spent on the project's phases.

System Timeline		
Phase	Tasks	Days Allocated
Inception	User Requirements	3
	System Requirements	3
	Specification Detailing	1
Elaboration	Finish Brainstorming Design/Architecture	5
	System Analysis	7
	Structure Design	7
Construction	Code Website/App, Login System, Interactive Park Map	9
	Code Sensors, Alerts and Switch Logic	9
	Complete Database Connectivity for Catalog & Park Rangers	9
	Testing/Documenting	8
	Prototype → Release	3
Transition	Receive Feedback, Finish Testing	4
	Complete System Design Structure	5
	Train Users on System	7 (Concurrent)
	Present Final Product	
	Total Number of Days:	80

Section 2.10 - CIA Categorization

The CIA Triad is an important model in computer science that deals with information security; this model is designed to oversee and evaluate how a company or organization handles data when it is stored, processed, and transmitted. This model should not be confused with the CIA (Central Intelligence Agency), as they have no relation to one another. The CIA triad has 3 critical components, each representing an important aspect of information security. The 3 components are: confidentiality, integrity, and availability. Confidentiality means that an organization's data should be kept private, only authorized users and processes should be able to access or modify data. Integrity means that data should be able to be trusted and should be kept in a good state so that it may not be tampered with; a computer system that has integrity should be correct, authentic, and reliable. Availability means that data should be available to authorized users upon legitimate request; unauthorized parties must not have access to a company or organization's data, but authorized parties should have unimpeded access to that data when it is required. There are a variety of different problems that pose a threat to the security of an organization's data and to the CIA triad's goal such as backdoor attacks, denial-of-service attacks, direct-access attacks, eavesdropping, privilege escalation, phishing, social engineering, spoofing, and tampering to name a few. All of these attacks mentioned are potential dangers designed to exploit, steal, change, and delete the data of a system for the hacker's benefit. One of the most common attacks in cyber security is social engineering. This happens when a user is tricked by a website they deeply trust, but has been hacked, into running a Trojan Horse virus on their computer; the user is then unaware that the compromised website delivers malware instead of the normal source code. As a result, the user may accidentally install malware that ends up stealing or compromising sensitive information, which can lead to the hacker taking complete control over the user's computer. A countermeasure to this attack would be to educate our users on today's editing threats in cyber security; people cannot be babysitted, so they must be constantly educated on the dangers that exist in computer science. Another way would be to install an anti-malware software, but user education would be most efficient as they would be able to spot threats a mile away. Another common attack in cyber security is phishing. Research suggests that approximately 50-60% of emails are spam, and many of them are phishing attacks that use valid credentials from legitimate entities in order to trick users into thinking the emails are legit, thus resulting in the user giving the hacker their important logon credentials. A countermeasure to this attack would be to install 2-factor authentication in the event that a user accidentally shares personal information after being tricked, and of course constantly educating users on how to spot fake emails that look like they are from legitimate organizations. Another common attack in cyber security is backdoor hacking. Backdoor hacking occurs when a hacker is able to bypass normal authentication processes and security protocols. A countermeasure to this attack would be installing a firewall or an antivirus software in order to detect when someone on the outside is trying to compromise the system from the inside. Another countermeasure would be patching our software and constantly updating it to meet current industry standards, as unpatched software usually exposes vulnerabilities that lead to backdoor attacks on systems.

—LIFE-CYCLE MODEL [3.0]—

Section 3.1 - Model used for Project

During the development process of our software system, we were faced with the decision of choosing which Life-Cycle Model to incorporate in order to guide us in designing the system. The decision came down between two choices: The Waterfall Model and the Rational Unified Process Model. Ultimately, we chose the Rational Unified Process Model (RUP). We chose this model because we believed that it was the choice that best fitted our vision for the project. The Rational Unified Process Model is a bit more complex than other simpler-to-use models, and we decided to use it for that reason because we wanted those extra details along the way.

Section 3.2 - Development Process

Section 3.2a - Pros of Model during Development

One of the positive aspects of using the Rational Unified Process Model (RUP) was that it emphasized the need for accurate documentation, allowing us to divide our work into phases. The phases were Inception, Elaboration, Construction, and Transition. The Inception Phases consisted of two Process Workflows: Business Modeling and Requirements. This phase is where we spent time gathering both user requirements and system requirements to properly brainstorm ideas about what we want the system to look like, who the system will benefit, and if profits will be made off of it. Next, the Elaboration Phase consisted of 3 Process Workflows: Business Modeling, Requirements, and Analysis & Design. This phase carries on the work started by the first phase, where we finish modeling and gathering requirements; however, we completed an additional step of analyzing how our system will be designed in order to perform the desired functions. Next, the Construction Phase consisted of two Process Workflows: Implementation and Testing. This phase is where we started the construction of our software and where we began integration of the development process. We also began to run validation and verification tests on areas of software already completed in order to determine if they met the requirements specified by the client. Lastly, we released our first prototype to be tested. Finally, the Transition Phase consisted of two Process Workflows: Testing and Deployment. In this last phase we received feedback on our prototype in order to finish running all tests on our system, then we determined the final design structure of our software system, and lastly we deployed our final product to the client. Another positive aspect of using this model was that it gave us a visual on how much time each phase should take to complete, allowing us to allocate resources more efficiently. This was useful for enhancing team productivity and helping reduce unexpected development costs. As we can see, the Construction Phase took the most time and resources to complete, followed by Elaboration, Transition, and Inception. Another positive aspect of using this model was that it allowed for a more iterative process that embraced incremental releases of the product. With this more iterative approach, we were able to explore different ideas throughout each phase based on stakeholder feedback, improving process control and risk management. In addition, it allowed us to deal with changing requirements, whether they came from the client or project itself.

Section 3.2b - Cons of Model during Development

As beneficial as the Rational Unified Process (RUP) Life-Cycle Model can be, it has its negative aspects as well. One of these negative aspects is that even with unfathomable resources, the RUP model is more complex and can be difficult to understand, leading team members to become disorganized. Thankfully we were able to stay on track, but there were times when the project got overly-complicated due to the development process, which caused a lot of stress among group members. Another negative aspect of the Rational Unified Process Model is that process details are expressed in general terms, providing minimal guidance and requiring us to use local customization in order to apply the model for our project's needs. This leads to another problem: since the model is reasonably complicated, developers must be experts in their field in order to properly develop software under this methodology. We had to adapt certain phases to fit the scope of our project, it was not an easy process and required more research which slowed down our development process. Another negative aspect of this model is that heavy documentation can become very expensive. Nothing is free in this world, especially when dealing with technology that involves valuable metals and materials. Allocating resources was one of the biggest challenges we had, with budgeting being one of the highest priorities. Constantly revising software systems to document them for user manuals can cause a strain on resources, thus we had to do our best to limit redundant documenting. In addition to straining resources, another negative aspect of using this model is that heavy documentation can be too slow for certain projects that need to be completed faster, and sometimes the model relies too much on stakeholder feedback to move forward efficiently. For projects utilizing new technology, components can't be reused, hence time spent on development can't be saved. Since we had more time to complete our project, we did not encounter this problem. However, even an iterative process might be too slow for other developers looking to speed things up.

Section 3.3 - Improvements on Model

As stated before, while the Rational Unified Process (RUP) Model is extremely beneficial for those who need it and know how to utilize it, it isn't perfect and could use some improvements. One of the improvements that I suggest is for the RUP model to be more useful and adaptable for smaller projects and teams. While our project was large in scale, our team was relatively small in size compared. As a result we ran into implementation problems during the construction phase of our software system, which caused delays and interrupted workflows. The RUP Model is great for large teams and large projects, but can be difficult to implement when dealing with smaller sizes and scales. On the other hand, another improvement I suggest is for the RUP model to focus less on integration during certain phases of the project. The reason for this is because one of the disadvantages of the RUP Model is that it emphasizes the integration of modules throughout the software development process, resulting in an adverse effect on more fundamental activities during the stages of testing. Integration throughout the process of software development sounds good in theory, but on large-scale projects with multiple development teams it will only add to the confusion, especially during testing.