Mountain Lion Detection Software System

Software Design Report

Turner Trowbridge

December 9, 2022

Table Of Contents

Section 1

User Manual

1.1. System Overview

This document covers the software design specifications for a mountain lion detection software system for the San Diego County Parks and Recreation Services for use in their parks. This software is being developed to notify a park ranger when and where a mountain lion is detected inside the park. This software system will utilize noise detectors that have been strategically placed around the park in a 5-square-mile radius around the central radio station to detect animal noises and notify the rangers.

1.2. Features and Functions

1.2.1 User Accounts

Each user on the system will need to have an account set up that contains basic information and allows them to sign into the system. The user accounts allow the system to log a user's activity.

1.2.2. Alert and Alarm Customization

The user of this system will be able to change the sensitivity settings for the noise detectors to push alerts. The user will also be able to change the sound the alarm plays and the cooldown for the alarm.

1.2.3. Active Monitoring, Alerts, and Alarm

The mountain lion detection software system will be actively monitoring all the noise detectors that have been set up around the park. When an alert meets the user's inputted sensitivity setting, an alarm will sound, which will display information about the location of the alert and will prompt the user to classify the alert as d*efinite, suspected, or false*. Once an alert has been responded to, an alert will be created for the same noise detector during the user-specified cooldown.

1.2.4. Saved Alert Data

When an alert has been classified by a ranger, the alert classification will be saved alongside the location, date & time, and the ranger who classified the alert. This will then be saved in an online database and stored for up to 30 days. Once a saved alert has been for over 30 days, the system will automatically create or add the alert to a less detailed summary of alerts in the past year.

1.2.5. Reports

Utilizing the saved alert data, a user will be able to request an extremely detailed report of activity in a specific time period within the last 30 days. These can be in the form of a report by date, classification, detector location, ranger, or a graphical report.

1.3. User Requirements

The user has specified that they want the mountain lion detection system to allow rangers to receive alerts on where animal noises are detected throughout the park. They want noise detectors set up in a 5-square-mile radius around the park.

The user has specified that they want the rangers to be able to change the noise level sensitivity of the noise detectors which would set off alerts. The user also wants to be able to receive alert messages with detailed information about where the sound occurred (within a 3-meter range), what type of noise was detected, and the strength of the noise detected. The alerts will sound off an alarm when received and will only turn off when a ranger responds to the alert, and classifies the alert as *definite, suspected, or false*. Once classified, another alert will not sound again until after a set time frame. The system will save all alerts received in the past 30 days. Once 30 days have passed, it will save a summary of these alerts for the year. Lastly, the user would like to be able to request reports based on these saved alerts. The requested types of reports are a report by date detected, classification, location, ranger, and a graphical report.

Section 2

Software Design Documents

2.1. Software Requirements

2.1.1. Functional Requirements

For the software to meet the user's requirements, on the front end, it needs to have a UI that will allow the user to interact with the software. The UI will let the user log in, allow the user to change the detection sensitivity, show alerts, play the alarm, let the user classify the alert as *definite, suspected, or false*, and let the user request reports.

On the back end of the system, the system needs to be able to constantly be receiving input from motion detectors. If a motion detector picks up a noise that meets the user's set sensitivity, an alert needs to be created with the motion detector's data and an alarm needs to be sounded. Once, the user classifies the alert, all the relevant information, such as the detector data, user data, and user input data needs to be saved to the database. This database also needs to be backed up online every day to ensure that a power outage or disk does not cause the saved data to be lost. When the user requests an alert, the system needs to be able to access the data from the online database and generate a report. Lastly, the database needs to remove any saved alert data older than 30 days and generate a summary based on that data in another part of the database.

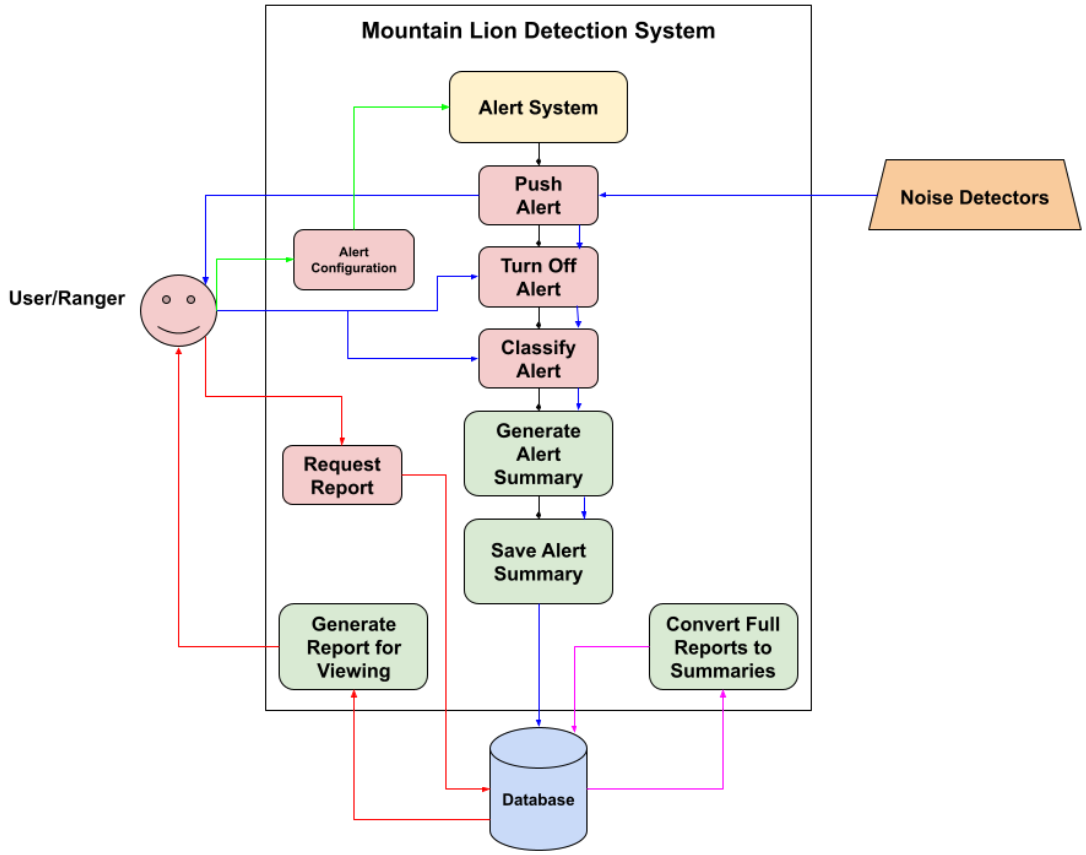
2.1.2. Non-functional Requirements

The software will need to be able to accurately get the data from the motion sensors in all types of weather. These motion sensors will need to be spread out in a 5-square-mile radius. There will need to be a smooth and quick interaction between the motion detectors and

the system that is run on the controlling computer in the ranger system in consideration of ranger and civilian safety. The system should be able to securely store the data collected and store it in the database. Due to the system running on the controlling computer in the parks and recreation center, the system could go completely offline during internet and power outages, therefore the data should be backed up online. Wear and tear on the motion sensors might affect the frequency at which the sensor sends out alarms and should be taken into consideration.

2.2. Use Cases

2.2.1. Use Case Diagram



2.2.2. Use Case Descriptions

Above is the mountain lion detection system use case diagram. This diagram showcases various use cases that can happen during the

normal operation of the system. It shows what interactions will occur between the mountain lion sensor, park ranger, and database.

Use Case 1 (Green Arrows)

The park ranger can configure how they want the system to alert them. Here, they can program the system to detect various types of animal noise. They can adjust the strength of motion they want to pick up to exclude events such as tree movement or small animals.

Use Case 2 (Blue Arrows)

This is the interaction between the mountain lion detection system, alert system, and the user. In this use case, the mountain lion detection system will pick up motion and send the information to the alert part of the software system. This will include the noise detected, strength, and the location of the detected noise within 3 meters. Once the alert system receives the data, it will set off an alarm on the controlling computer, alerting the park rangers. This alarm will sound until a ranger turns it off. At this time, the ranger will be required to classify this alert as either “definite”, “suspected”, or “false”. The alarm will then not sound if the same motion detector picks up movement within the next 30 minutes. The system will then save all the data collected, including the data from the motion sensor, the reliability of the alert set by the park ranger, and the ranger's name to the database.

Use Case 3 (Red Arrows)

This use case showcases the park ranger requesting and viewing a report. The park ranger can request this report from the software. The software will utilize the alert summaries to generate a report based on the park ranger’s selection and present it to the ranger for viewing.

Use Case 4 (Pink Arrows)

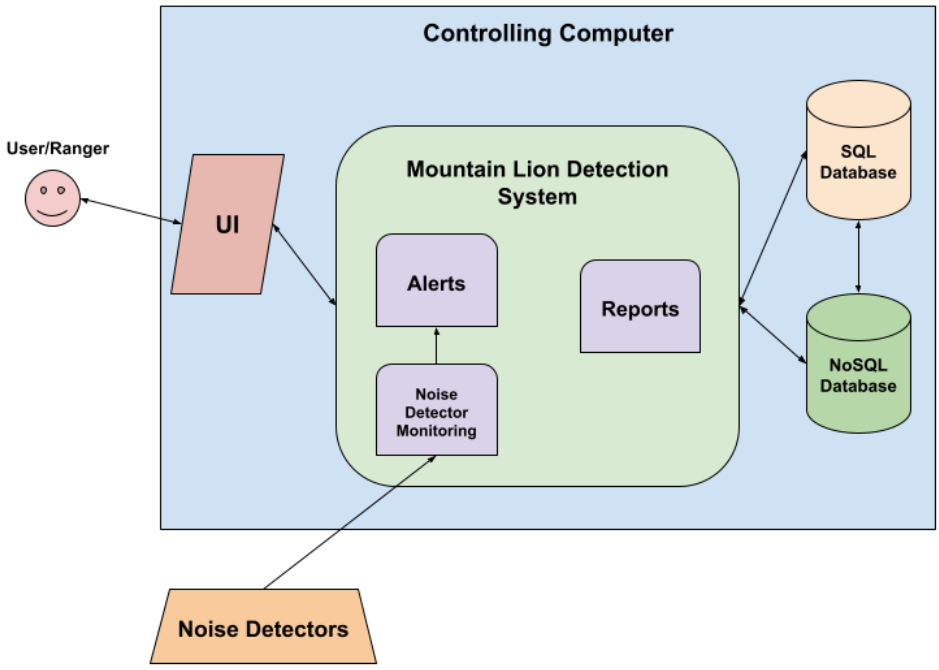
The system will automatically convert any data older than 30 days into a brief summary of the past year. This will be done in an interaction between the database and the software.

2.3. Validation

The use case diagram above accounts for all the user's requirements, including both functional and non-functional requirements. The user has specified that they want a system that allows the user to change the sensitivity of the alarms, which is what the first use case accounts for. The second use case lays out what the client has specified for the alert system. The interaction between the noise detector and system, and the user and system, all highlight these requirements. These interactions showcase how the user will be notified when a noise is detected above the set level and show how the user will be required to classify the alert. Lastly, the third use case showcases the user's requirement for creating a report based on the saved report data, and the creation of a summary of data older than 30 days.

2.4. System Architecture

2.4.1. Architecture Diagram



2.4.2. Architecture Description

The architecture of the system is based around the controlling computer. A UI will be displayed to the user which will allow them to interact with the system. The system itself will consist of a way to take in input from the noise detectors and actively monitor this input. It will also contain the alert and alarm implementations, which will notify the user via the UI when a noise above the user-inputted threshold has been detected. The user will be able to change various settings within the system via the UI. When a user classifies an alert, the alert will be saved into the NoSQL database. Records of the user's information will be stored inside the SQL database. Lastly, the UI will allow the user to interact with the reports, which will be generated based on the NoSQL database.

2.5. Security

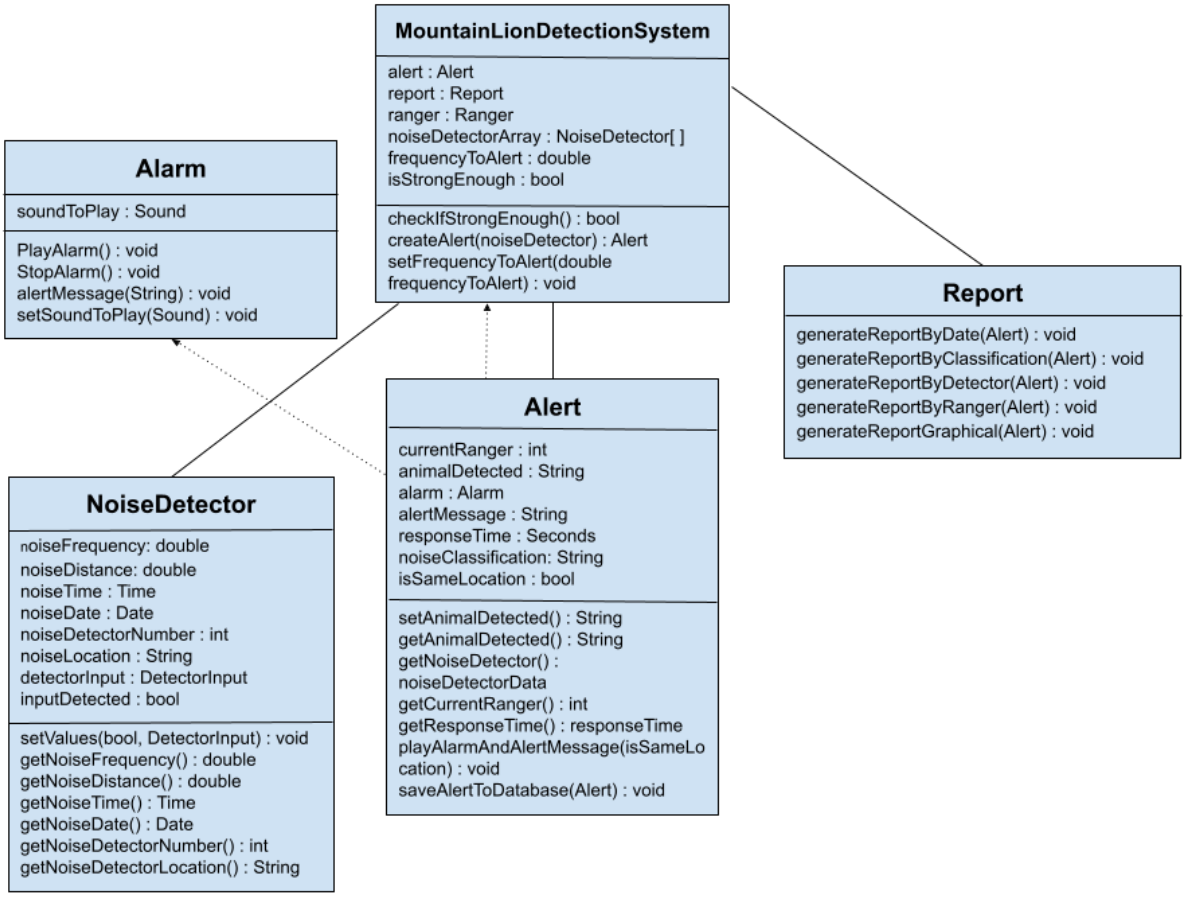
Basic security measures should be in place on the controlling computer to prevent an unknown user from accessing the server. This should include each user having their own account and password to log into the computer and a separate account and password to log into the Mountain Lion Detection system. This ensures that the confidentiality of the system is not violated and reduces the risk of backdoor entry into the system.

To ensure the integrity of the system measures should be taken that verify that data within acceptable ranges is saved to the database. This will prevent invalid data from being added to the system. The database should also only accept entries from logged-in rangers on the controlling system. With the database storing user logs, it should be easy to track down and remove entries that have been added if a user's information was stolen by something like phishing or social engineering. In addition, security cameras should be set up inside the ranger system to monitor the controlling computer.

The system should operate on its own private network to prevent tampering and eavesdropping to ensure availability and confidentiality. The motion detectors will be connected through the private network, allowing no access from outside users. The saved data will then be uploaded to a database that is secured by a third party to add additional monitoring that does not require the San Diego Parks and Recreation to hire experts in this field.

2.6. System Classes

2.6.1. Class Diagram



2.6.2. Class Descriptions

MountainLionDectectionSystem Class

The MountainLionDetectionSystem class allows the user to access various parts of the system. This class is responsible for creating an array of all the noiseDetectors that have been set up around the park. The user will be able to set the frequency at which they want an alert to happen. With this set frequency, while the noiseDetector receives input, it will call checkIfStrong() and return a boolean. If this boolean is true an alert will be created using createAlert(noiseDetector).

Alert Class

The Alert class will be responsible for creating an alert that will sound an alarm using the Alarm class and output an alertMessage. The system will prompt the user to respond using a string and track the

number of seconds it takes to respond. Using the currentRanger, it will also be able to track which ranger was responsible for responding to the alarm. The isSameLocation boolean will be used as an input when trying to playAlarmAndAlertMessage per the user's requirements that a new noise sensor will need to have gone off. Lastly, the alert class will then call saveAlertToDatabase passing an instance of Alert.

Alarm Class

The Alarm class plays a sound that is used when the PlayAlarm() method is called and outputs an alarmMessage at the same time. The alarm will stop playing when the StopAlarm() method is called. Additionally, a setSoundToPlay method allows the user to change the alert sound.

Report Class

The Report class will be able to generate reports using the data from the Alert and Past Alert Summary databases. These report types are are a report by date, classification, detector, ranger, and graphical report. They are called through their functions and will output the generated report on the screen.

Noise Detector Class

The NoiseDetector class will contain the noiseFrequency, noiseDistance, noiseTime and noiseData that are constantly being updated using a previously developed detection system, DetectorInput. This class will also log the noiseDetectorNumber and noiseLocation. All of these variables will then have a getter that will allow the MountainLionDetectionSystem to createAlert().

2.7. Verification

2.7.1. Unit Testing

Test Case: Alarm Class

- setSoundToPlay(sound1);
- PlaySound();

Expected Output: Sound1 is playing through the computer speakers.

Description: In this test case, the ranger will select a sound to play and attempt to play it, which will cause the sound to play through the controlling computer's speakers.

Test Case: Ranger Class

- ranger2.setName("Bob");
- ranger2.getName();

Expected Output: Bob

Description: This test is to check that the ranger name can be assigned correctly, by setting a ranger name to "Bob". Then the get method will be called to ensure the name has been set properly.

2.7.2. Functional Testing

Test Case: Generate a report based on an added alert.

- savedAlerts.addAlert(alert0);
- generateReport.generateReportByDate(savedAlerts.savedAlerts Array[0]);

Expected Output: Report that is sorted by date

Description: In this test, the generateReportByDate method is tested. To do this, an alert object with predefined values is added to the savedAlerts array in the first position. The generateReportByDate method is then called passing in this value that was just added to the first position in the savedAlerts array. The output will have the same values as the preset alert sorted by their dates.

Test Case: Alert message displays mountain lion.

- alert.setAnimalDetected("Mountain Lion");
- alarm.alertMessage();

Expected Output: Message shows up on screen alerting that a

mountain lion has been detected.

Description: In this test, the alertMessage method is tested to see if it will display the correct animal that has been detected. To test this, the alert object's animalDetected string is set with "Mountain Lion". Next, an alarm object is called and creates an alertMessage using the string that was set. An alert message should now pop up on the controlling computer's screen with "Mountain Lion".

2.7.3. System Testing

Test Case: Alarm and alert message are set off when noise is detected.

This system test is designed to see if the alarm and alert message are working properly. The test starts by setting up a noise detector in the intended location. Then, a value is set for the sensitivity of the noise on the controlling computer, such as 5.00. Next, a person will walk in the range of the detector, which picks up input that is continuously fed to the Mountain Lion System class. A method inside the class will constantly be checking if the detected noise input is equal to or greater than the noise sensitivity that was just set. A person walking in the range of the noise detector will meet this requirement, and thus an alert should be created based on the noise detector data. This should then set off an alarm and the alert message on the controlling computer, giving the expected output.

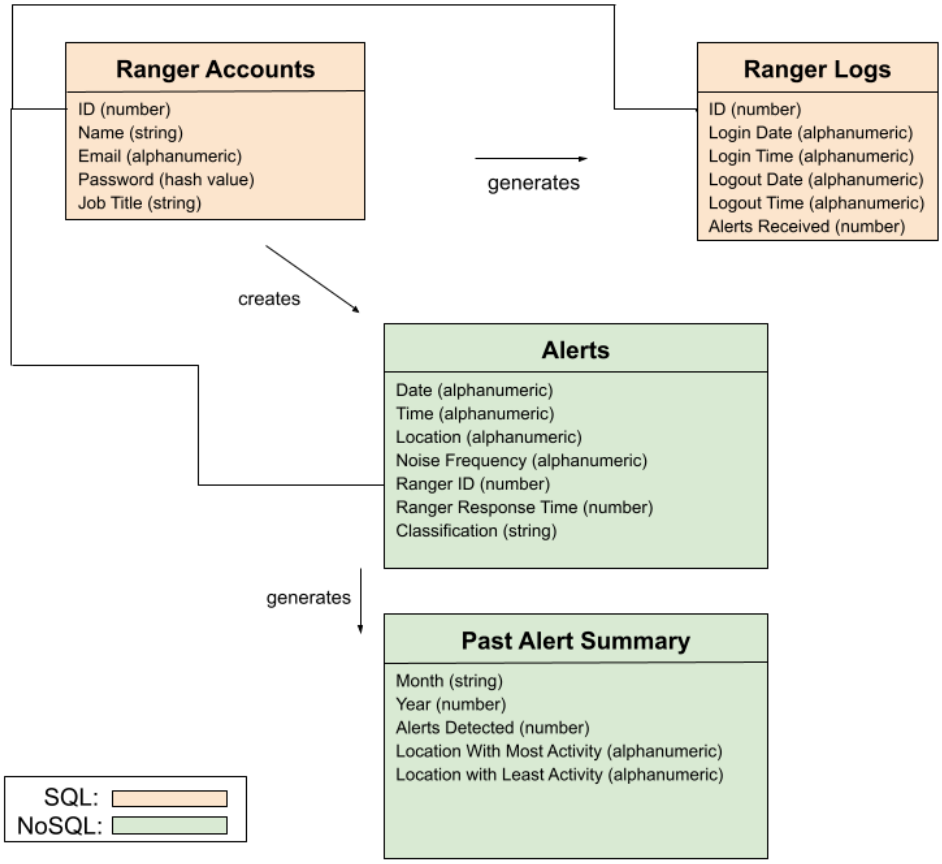
Test case: Ranger requests a report sorted by date.

In this test case, a ranger will log onto the system to request a report by imputing their name. This will then allow them to access the system where they will see some options that allow them to change settings for the system as well as the report request feature. The ranger will select "generate report" and will be promoted to select the type of report they want. For this test, the ranger will select a report by the date. The report will be generated by using the stored data in the SavedAlerts class and generateReportByDate method inside

GenerateReport class. This will then output a spreadsheet that shows the reports sorted by date.

2.8. Database Design

2.8.1. Database Diagram



2.8.1 Data Dictionary

Data Dictionary

Ranger Accounts

Term: ID
Type: Numeric
Description: A ranger's unique identification number.
Example: 328743

Term: Name
Type: String
Description: A ranger's full name.
Example: Bob Meyers

Term: Email
Type: Alphanumeric
Description: A ranger's email address.
Example: bob.meyers@sandiegoparks.gov

Term: Password
Type: Alphanumeric (hash value)
Description: The ranger's password is stored as an SHA-256 hash
Example: 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8

Term: Job Title
Type: String
Description: The specific job title of the ranger.
Example: Chief Ranger

Ranger Logs

Term: ID
Type: Numeric
Description: A ranger's unique identification number.
Example: 178763

Term: Login Date
Type: Alphanumeric
Description: Date ranger logged in.
Example: 8/12/22

Term: Login Time
Type: Alphanumeric
Description: Time ranger logged in.
Example: 9:45 AM

Term: Logout Date
Type: Alphanumeric
Description: Date ranger logged out.
Example: 6/27/22

Term: Logout time
Type: Alphanumeric
Description: Time ranger logged in.
Example: 6:58 PM

Term: Alerts Received
Type: number
Description: Number of alerts received during a ranger's session.
Example: 12

Alerts

Term: Date
Type: Alphanumeric
Description: Date of alert creation.
Example: 11/10/22

Term: Time
Type: Alphanumeric
Description: Time of alert creation.
Example: 7:45

Term: Detector ID
Type: Alphanumeric
Description: The unique ID of the noise detector responsible for alert creation.
Example: 045

Term: Noise Frequency
Type: Numeric (dB)
Description: Frequency of the noise detected that triggered alert creation.
Example: 60

Term: Ranger ID
Type: Numeric
Description: A ranger's unique identification number.
Example: 745699

Term: Ranger Response Time
Type: Alphanumeric
Description: Time elapsed before ranger responded.
Example: 40 seconds

Term: Classification
Type: String
Description: What the ranger classified the alert as.
Example: Definite

Past Alert Summary

Term: Month
Type: String
Description: Month of alert summary.
Example: October

Term: Year
Type: Numeric
Description: Year of alert summary.
Example: 2022

Term: Alerts Detected
Type: Numeric
Description: Number of alerts detected in month.
Example: 87

Term: Amount of Definite Alerts
Type: Numeric
Description: Number of definite alerts classified in month.
Example: 20

Term: Amount of Suspected Alerts
Type: Numeric
Description: Number of suspected alerts classified in month.
Example: 47

Term: Amounts of False Alerts
Type: Numeric
Description: Number of false alerts classified in month.
Example: 20

Term: Detector with Most Activity
Type: Numeric
Description: ID of detector with most activity in month.
Example: 081

Term: Detector with Least Activity
Type: Numeric
Description: ID of detector with least activity in month.
Example: 158

2.8.2. Database Justification

Ranger Accounts

An SQL database was chosen for the ranger account information due to an account only being created once per ranger. The most likely updates being made would be to delete a ranger from the system or if a ranger is changing their password.

Ranger Logs

An SQL database was chosen for ranger logs because the fields are fixed and no new fields will be added. A NoSQL database would be useful if there were multiple computers with different rangers logging in at the same time, however, our system is using only one controlling computer so an SQL database will work just fine.

Alerts

This database was chosen to be a NoSQL database because even though one ranger will be creating the alerts when they respond to alarms, these fields might want to be changed in the future to compensate for new technology. Also, there will be some sort of automation between this database and the Past Alert Summary database that will automatically create a summary passed on the data in the Alert database every month - storing it in the Past Alert Summary database - and then will delete the data from the Alert database.

Past Alert Summary

This database was chosen as a NoSQL database due to its very close interactions with the data fields in the other NoSQL database, the Alert database. While a SQL database would work fine, it will be nice to have the flexibility that a NoSQL database has when interacting so closely with another database that might frequently change its data.

2.9. Future Features

One possible future feature would be the ability to allow rangers to choose what animal has been detected. To implement this, a security camera would need to be installed at each noise detector to allow the ranger to see what animal is in the radius.

2.10. Timeline

The overall expected deadline for the Mountain Lion Detection System is 9 months. In these 9 months, specific deadlines for each member will be assigned. The first priority is a working prototype within 6 months. This prototype will be based on the class diagram and ensure that the system is able to run. Next, the prototype will be demonstrated to the clients to get feedback to ensure the system is in line with their requirements. Another development phase will then take place for another 2 months to add these new requirements. Lastly, a final test will take place to ensure the program is working correctly. This will involve field tests that will take about 1 month to complete and will result in a final build that will be ready for use at the recreational park.

Section 3

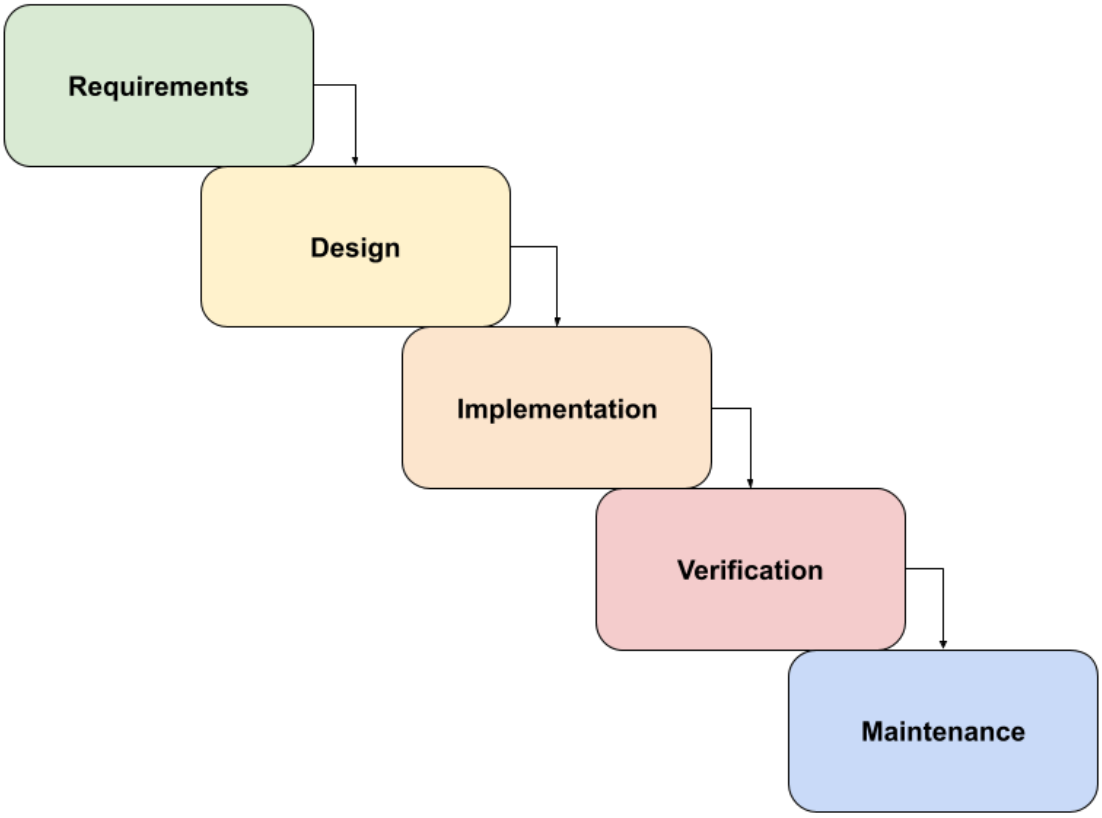
Life Cycle Model

3.1. Overview

The life-cycle model that will be utilized for this system is the waterfall model. The reason why the waterfall model will be utilized is because the system requested by the San Diego County Parks and Recreation Department is very straightforward. Additionally, due to the smaller scale of the project, the waterfall model will be just adequate.

3.2. Waterfall Model

3.2.1. Waterfall Model Diagram



3.2.2. Phase Descriptions

Requirements

In this phase, the user requirements, functional requirements, and non-functional requirements will be documented.

Design

In this phase, a use case diagram, an architecture diagram, and a class diagram will be designed to document the way the system will be developed.

Implementation

In this phase, the backbone of the system will be made based on the documents made in the design phase.

Verification

In this phase, the prototype of the system will test based on the verification testing cases. Additionally, a small-scale version of the system will be set up to demonstrate to the client.

Maintenance

Based on the client's feedback on the small-scale version of the system, small improvements and changes will be made before the system is fully set up and deployed in the field. Additionally, small improvements will be made in this phase based on bugs that are discovered during operation in the field.

3.3. Reflection

3.3.1. Benefits

The inherent benefit of the waterfall model is the early emphasis on the system requirements. Due to the user requirements being specific and clear, the software can be designed using these in a short amount of time. Therefore, once the requirements have been defined they will serve as the backbone of the system. When the working prototype is presented to the clients after 6 months, the system should be nearly done. Due to the initial requirements creating a solid backbone of the system, if the user has any new requirements, they will most likely be additions to the existing system and not include any large changes to the backbone.

3.3.2. Drawbacks

One of the main drawbacks when using the waterfall model is that there is no backtracking. This means that once the foundation has been set if the client wants to make big changes, the system might need to be rebuilt from scratch.

3.3.3. Improvements

One improvement that could be made to the life-cycle model that will be in this system, the waterfall model, is to report the design to the clients before the system is implemented. Also, the implementation phase could be split up, to allow updates to be added to the original design documents that can be shared with the clients to make sure the system is being set up the way the client wants it.