Software Design Specification: Mountain Lion Detection System

Prepared by Christian Byars, Nuonnettra Kanzaki, and Kyle Camposano

11th October 2024

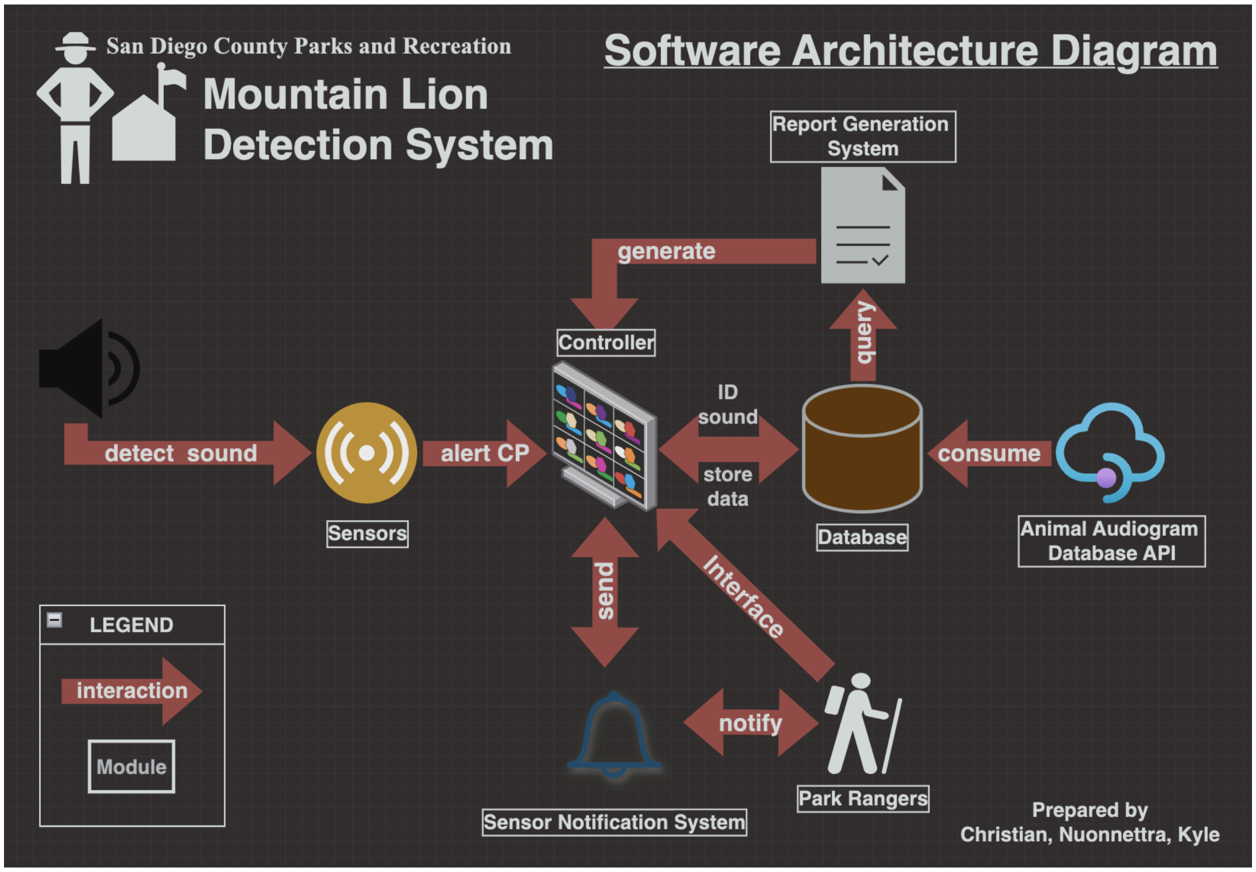
1. System Description

The Mountain Lion Detection System (MLDS) is dedicated to its hikers and park employees to detect possible dangerous animals on the mountain trails and to notify nearby recreational hikers. The system consists of detection sensors covering up to 5 square miles to pinpoint the approximate location of the animals within 3 meters. Upon detection of mountain lions, the system notifies park rangers, stores all mountain alerts in local storage for up to thirty days, and stores a summary of alert information for up to one year. The park rangers utilize the best course of action to notify nearby hikers of the mountain lion threat within the proximity. The system's control program allows park rangers to access the stored data at any time, which can then be used to generate reports on the detections of mountain lions. The development of the Mountain Lion Detection System is crucial for the safety of the general public and park employees at San Diego County Parks and Recreation and possibly other California State Parks. This document is separated into four categories: software architecture diagram and its descriptions, UML class diagram and descriptions, partitioning of tasks in the team, and the project timeline.

2. Software Architecture Overview

This section illustrates how the software works. The general flow is that the sensors would detect a noise that can be categorized as a mountain lion. This detection is then sent to the controller which generates a notification to alert the park rangers of the potential danger. The database will take every data into account to make a report which is then sent to the controller where the user or park ranger has access to display or view the report. The user or park ranger would then decide what the manual course of action would be by interaction with the controller such as going through different report types or manually putting in a classification (suspected, definite, or false) for that report. This section will then lead to the description which includes the description and interaction of all the six modules, Sensors, control program, Report Generation System, Sensor Notification System, Park Ranger, Database, and Animal Audiogram Database API.

2.1 Software Architecture Diagram


San Diego County Parks and Recreation
Mountain Lion Detection System
Software Architecture Diagram
Report Generation System
generate
query
Controller
detect sound
Sensors
alert CP
ID sound
store data
Database
consume
Animal Audiogram Database API
send
Interface
LEGEND
interaction
Module
notify
Sensor Notification System
Park Rangers
Prepared by
Christian, Nuonnettra, Kyle

2.2 Descriptions of Software Architecture Diagram

1. **Module:** Sensors
   a. **Description:** The sensors are responsible for detecting animal sounds immediately and accurately at their designated location. Each sensor operates in a 5 square mile radius and pinpoints the sound's approximate location within 3 meters. The sensors are placed throughout the park near the hiking trails to ensure the safety of the hikers.
   b. **Interaction:** The sensor is event-driven and is only triggered when animal sounds are detected. The sensor then relays this information and alerts the Control Program to classify the sound, notify the hikers, and store the sound data.

2. **Module:** Sensor Notification System
    a. **Description:** The sensor notification system is responsible for making the possible threat of mountain lions apparent. This is done by sounding off an alarm once an alert message is received.
    b. **Interaction:** The Sensor Notification System will sound off the alarm through the controller once it has the appropriate data. This will then notify park rangers about the threat of one or multiple mountain lions.

3. **Module:** Animal Audiogram Database API
    a. **Description:** The Animal Audiogram Database contains data regarding the sounds of various animals. This allows the system to accurately detect mountain lions and not the sounds of other animals.
    b. **Interaction:** The audiogram database feeds its data to the primary database.

4. **Module:** Report Generation System
    a. **Description:** The Report Generation System provides written documentation of various data the detection system picks up. This includes reports showing all mountain lion detections, the sensor triggered, a graphical report of detections within two miles of the park, and detection classifications by ranger.
    b. **Interaction:** The report generation system needs the data from the database in order to provide reports. Once the reports are generated, they will be accessible via the control program.

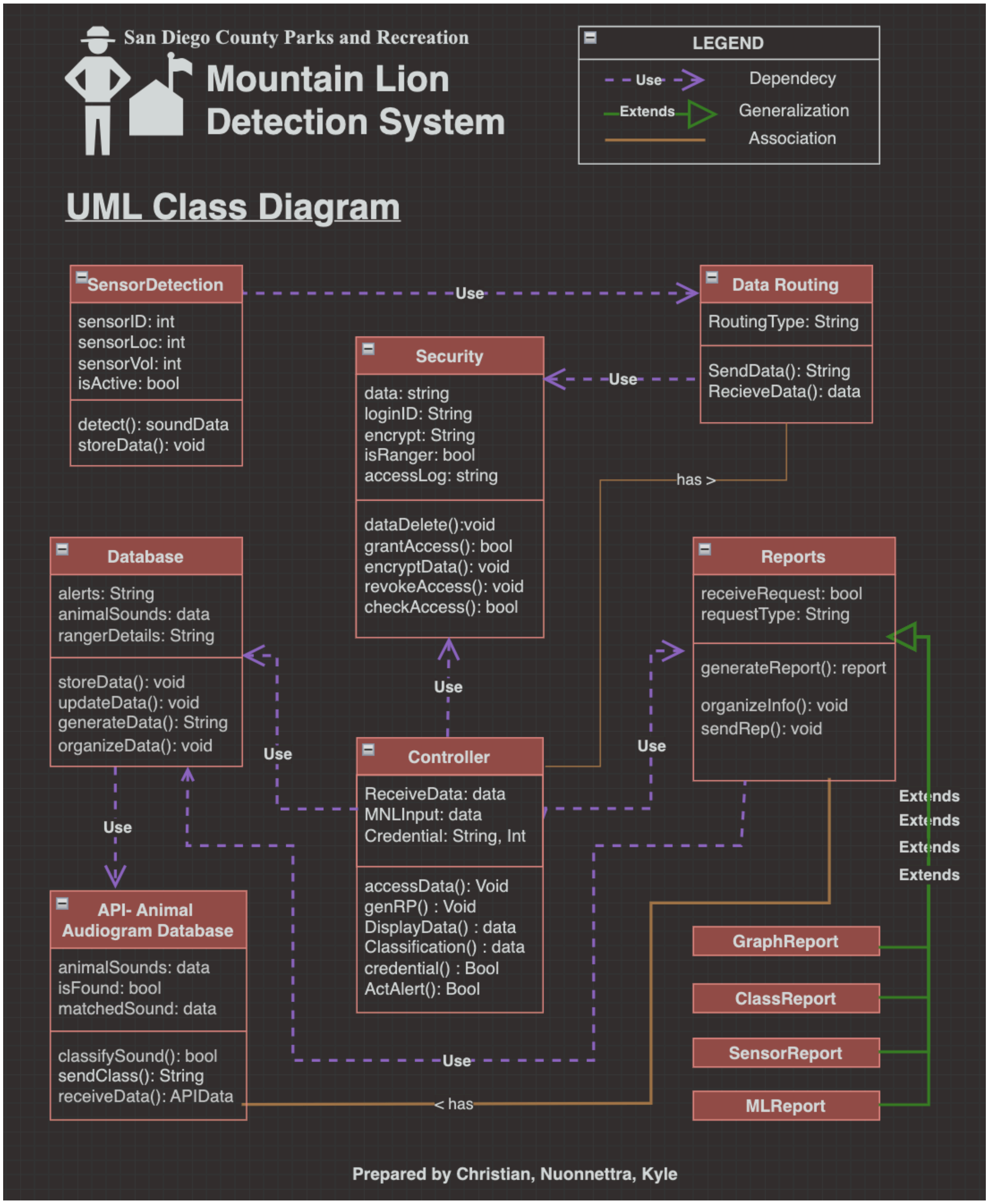
5. **Module:** Controller
    a. **Description:** The controller is the most crucial component of the software architecture. It is where most of the information and interaction go. This is where the data about the possible mountain lion will be sent, which in turn, the control program will send to the database. The controller is also where the park ranger can interact to use other components of the program. This includes the module, report generation system, database, and sensor notification system.
    b. **Interaction:** The park ranger can generate specific types of reports, view notifications, turn on or off the alarm system, and put in a classification (Definite, Suspected, or false) for a report.

6. **Module:** Database
    a. **Description:** This component is where all of the data is stored or will be stored. This is where all the history reports are, to which all the controller and the report generation system have access for their own functioning.
    b. **Interaction:** The database depends on the control program to receive the data and send the data back to the control program. This component will store according to the standard rule of which to have a full report for 30 days and a summary of the report for up to a year and then discard the reports once it passed the set time period.

7. **Module:** Park Ranger

a. **Description:** This is the verified user which is the authorized park rangers that will have control and access over the system.
b. **Interaction:** The user can go to the controller to access the database, the reports, and the alert notification system.

2.3 UML Class Diagram



2.4 Descriptions of UML Class Diagram

1) **Class**

SensorDetection: This class is a data structure holding all the information and operations of each sensor part of the software system. Each sensor can detect animal sounds, categorize them, and then send the data to the local data storage.

Attributes:

- sensorID: A numerical identifier for each sensor to ensure they're properly identified. This ID is unique and tied to the sensor.
- sensorLoc: The physical location of each sensor to ensure they're reliably located in the park in case of maintenance and adjustments.
- sensorVol: A numerical representation of each sensor's volume to ensure that hikers hear the warning alarms at a safe volume.
- isActive: A representation of each sensor's current status/operability.

Operations

- detect(): Detects animal sounds within the range of each sensor. Returns data as soundData.
- storeData(): Stores and categorizes the animal sounds to the local data storage.

2) **Class**

Database: This class is an interface representing all the information and operations of all the collected data from the sensors and control program. It handles all of the data about the animal sounds and alerts. This class interacts with the Class Controller, Class Reports, and Animal Audiogram Database API.

Attributes

- alerts: Collected alerts of detected mountain lions from each sensor in the park.
- animalSounds: Collected data of detected animal sounds from each sensor in the park.
- rangerDetails: Information of the ranger at the time of data collection, update, access, or retrieval.

Operations

- storeData(): Stores all collected data into the database from the control program and sensors.
- updateData(): Updates the data in the database which includes the animal type, animal information, and duration stored data in the database.
- generateData(): Generates the matching data that classifies the animal directly pulled from the API. Returns data as String.
- organizeData(): Organize the data according to their animal type and sounds.

3) **Class**

Animal Audiogram Database (API): This class holds all information about the animals and their respective sounds. It allows the mountain lions to be identified and classified with accuracy. It interacts with one of the major components of the MLDS, the database.

<u>Attributes</u>

- animalSounds: The collection of all data of animal information and sounds stored in the API.
- matchedSound: The animal data that matches the animal sound collected from the sensor.
- isFound: A status of whether the collected sound from the sensor does match a sound from the API.

<u>Operations</u>

- receiveData(): Receives the animal sounds from the sensors in the park. Returns data as APIData.
- classifySound(): Classifies the animal sounds pulled from the control program and then matches it with the animal sound in the API. Returns a boolean showing if the animal sound matches a classification.
- sendClass(): Sends the classification of the matched animal based on its sound to the control program. Returns String for confirmation.

4) <u>Class</u>

Reports: This class holds all information about report requirements and is the template interface for the Graphical Report, Classification Report, Sensor Report, and Mountain Lion Report. It interacts with the control program to generate the appropriate reports requested by the park rangers. This interacts with the Class Database (dependency) to fetch relevant information for the requested report.

Attributes

- receiveRequest: A status indicating whether a report is requested from the control program.
- reportType: The type of report being requested by the park ranger. (Graphical, Classification, Sensor, Mountain Lion)

Operations

- generateReport(): Generates the appropriate report based on the type of report. Returns a report data type that matches the class that represents the report.
- organizeInfo(): Organizes and compiles the appropriate information into a report.
- sendRep(): Sends completed, requested reports back to the control program.

5) **Class**

GraphReport: This is the graphical report showing detections on a map of the park and areas within 2 miles of the park. It extends the Class Reports including all of its attributes and operations. This interacts with the Class Reports to generate the graphical report for the control program.

Attributes

- receiveRequest: A status indicating whether a report is requested from the control program.

- reportType: The type of report being requested by the park ranger. (Graphical)

Operations

- generateReport(): Generates the appropriate report based on the type of report. Returns a report data type that matches the class that represents the report.
- organizeInfo(): Organizes and compiles the appropriate information into a report.
- sendRep(): Sends completed, requested reports back to the control program.

6) Class

ClassReport: This is the report showing detection classifications by the park rangers. It extends the Class Reports including all of its attributes and operations. This interacts with the Class Reports to generate the detection classification report for the control program.

Attributes

- receiveRequest: A status indicating whether a report is requested from the control program.
- reportType: The type of report being requested by the park ranger. (Classification)

Operations

- generateReport(): Generates the appropriate report based on the type of report. Returns a report data type that matches the class that represents the report.
- organizeInfo(): Organizes and compiles the appropriate information into a report.
- sendRep(): Sends completed, requested reports back to the control program.

7) Class

SensorReport: This is the report showing all mountain lion detections at a specific sensor location. It extends the Class Reports including all of its attributes and operations. This interacts with the Class Reports to generate the sensor report for the control program.

Attributes

- receiveRequest: A status indicating whether a report is requested from the control program.
- reportType: The type of report being requested by the park ranger. (Sensor)

Operations

- generateReport(): Generates the appropriate report based on the type of report. Returns a report data type that matches the class that represents the report.
- organizeInfo(): Organizes and compiles the appropriate information into a report.
- sendRep(): Sends completed, requested reports back to the control program.

8) Class

MLReport: This is the report that shows all mountain lion detections by date detected and by classification (definite, suspected, or false). It extends the Class Reports including all of its attributes and operations. This interacts with the Class Reports to generate the mountain lion detection report for the control program.

Attributes

- receiveRequest: A status indicating whether a report is requested from the control program.

- reportType: The type of report being requested by the park ranger. (Mountain Lion)

Operations

- generateReport(): Generates the appropriate report based on the type of report. Returns a report data type that matches the class that represents the report.
- organizeInfo(): Organizes and compiles the appropriate information into a report.
- sendRep(): Sends completed, requested reports back to the control program.

9) Class

Security: The security class ensures all data is encrypted and that data is only accessible by park rangers. This is used by the controller and data routing classes. The system also deletes data after a fixed number of days.

Attributes

- Data: The information collected by the detection system.
- loginID: A unique identifier that is assigned to each user of the system.
- encrypt: a form of data security in which the data that is collected is made only readable and accessible to park rangers.
- isRanger: checks to see if the individual who is requesting access is a park ranger.
- accessLog: creates a history of who logs into the system and the time at which it was accessed.

Operations

- dataDelete(): deletes data recorded by the detection system after a specific number of days.

- grantAccess(): gives access to newly authorized park rangers. Returns a boolean verifying the user is authorized to access the controller.
- encryptData(): Takes in the data that is received and encrypts it.
- revokeAccess(): removes the access of previously authorized users.
- checkAccess(): checks to see if the park ranger who wants to access the stored data is authorized. Returns a boolean to ensure the park ranger is verified.

10) Class

Controller: This class is responsible for displaying or delivering the report or data to the verified park ranger. It is also responsible for taking in manual commands from the verified ranger.

Attributes

- ReceiveData : This is to receive the data from the data routing into the controller.
- MNLInput: This is to give the verified ranger the ability to manually turn off the alarm or give classification(Definite, Suspected, False) for a report.
- Credential: This is where the verified ranger logs in to his or her account which requires an ID and a password.

Operations

- accessData(): access data from the data routing.
- genRp(): generate the report for the user to access.
- Classification(): let the user input the classification (Definite, Suspected, False) for each report that they have to deal with.

- DisplayData(): gives the user the option to visually view the data including a map. This also includes the next page and previous page. The parameter takes in the location of the data to generate the Visual report.
- Credential(): check the user credentials as to whether they are the verified rangers. The parameter will take in the ID and Password. The return type will be true or false to indicate whether the user is verified or not.
- ActAlert(): give the user access to the alarm to turn off or on. This method returns true or false reflecting the status of the alert.

11) Class

DataRouting: This class is responsible for delivering the data between each class and ensuring that the data is being sent with its original storage or quality.

Attributes

- RoutingType : This is to ensure the direction the data is being sent to or from.

Operations

- SendData(): an option for the routing to send. The parameter will take in the data to be sent.
- ReceiveData(): an option for the routing to receive. The parameter will take in the location or the address of the data to be received and then it will return the data.

3. Team Responsibilities and Timeline

All team members assumed equal responsibilities in the planning of the MLDS in order to complete the project throughout its expected lifespan. This section provides the breakdown of the team members' tasks/ responsibilities and timeline.

3.1 Team Members' Responsibilities

Christian Byars

- Created the security class along with its operations and attributes.
- Proofread report for spelling, grammatical, and formatting errors.
- Contributed to the descriptions and interactions for 3 modules of the software architecture diagram: Sensor Notification System, Animal Audiogram Database API, and Report Generation System.
- Contributed to filling out the group timeline.

Nuonnettra Kanzaki

- Involved with multiple classes called controller and data routing. Wrote the attributes and operations for those classes.
- Wrote software architecture overview
- Wrote description and interaction for the Software Architecture Diagrams modules for Controller, Park Ranger, and Database.
- Proof Check the Software Architecture Diagrams and User Case Diagram.

Kyle Camposano

- Responsible for designing the Software Architecture Diagram and creating the template for the UML class diagram. This also includes outlining the model for the sensor detection, Animal Audiogram Database API, and all required reports.
- Responsible for detailing the specifications of the software system, including providing descriptions of classes, attributes, and operations of the SensorDetection, Database, API, and Reports
- Responsible for implementing a feature to extend the base class report to generate specific reports (graphical, sensor, classification, and mountain lion detection)
- Responsible for creating an approximate timeline for the lifespan of the project, detailed weekly with specific tasks.

3.2 Timeline

Week	**Project Tasks**
1-2	Requirements Gathering, Basic Software Requirements Specification
3-5	Software Design Specification, UML Diagram
6-7	Client Conference, Developers Implementation
8-9	Finalize Development, Polish Details

Verification Test Plan: Mountain Lion Detection System

Prepared by Christian Byars, Nuonnettra Kanzaki, and Kyle Camposano

25 October 2024

1. System Description

The Mountain Lion Detection System (MLDS) is dedicated to its hikers and park employees to detect possible dangerous animals on the mountain trails and to notify nearby recreational hikers. The system consists of detection sensors covering up to 5 square miles to pinpoint the approximate location of the animals within 3 meters. Upon detection of mountain lions, the system notifies park rangers, stores all mountain alerts in local storage for up to thirty days, and stores a summary of alert information for up to one year. The park rangers utilize the best course of action to notify nearby hikers of the mountain lion threat within the proximity. The system's control program allows park rangers to access the stored data at any time, which can then be used to generate reports on the detections of mountain lions. The development of the Mountain Lion Detection System is crucial for the safety of the general public and park employees at San Diego County Parks and Recreation and possibly other California State Parks. This document is separated into five categories: test plan overview, feature 1 and its test cases, feature 2 and its test cases, partitioning of tasks in the team, and updates on the original Design Specification.

2. Test Plan for Verification

This section details the test plan for verification of the Mountain Lion Detection System. The test plan comprises two test sets focusing on two features, which totals three unit tests, two integration tests, and two system tests. It specifically verifies critical features in the software system. The test cases show their module independence/ dependency, interactions, and functionality. This section breaks down each test case with its targeted feature, description, controlled input(s), expected output, and overall steps of the test.

2.1 Feature 1: Alerting park rangers of mountain lion detection.

Unit Test for Feature 1

Feature: Alerting park rangers of the mountain lion detection.

Test Components: detect() in SensorDetection Class

Test Description: The purpose of this unit test is to verify that the detect() operation effectively and accurately detects animal sounds by utilizing a set of sounds. The operation returns a soundData object of the animal sound and is sent to the Controller to analyze and categorize it. The feature must detect any nearby sound. This test aims to cover the failure of not detecting nearby animal sounds.

Test Input: A set of sounds commonly produced/ heard at trails in San Diego County:

1) Mountain lions: growl
2) Cicadas: buzzing

3) Hikers: footsteps

Test Expected Output:

1) soundData with the raw sound of a mountain lion growl.
2) soundData with the raw sound of a cicada buzzing.
3) soundData with the raw sound of a hiker’s footsteps.

Unit Testing Steps

1) Input: growl from a mountain lion.

	Steps	**Input**	**Expected Output**
1	The sensor detects a nearby animal sound.	Sound detected: mountain lion growl.	soundData with the raw sound of a mountain lion growl.

2) Input: buzzing from a cicada.

	Steps	**Input**	**Expected Output**
1	The sensor detects a nearby animal sound.	Sound detected: cicada buzzing.	soundData with the raw sound of a cicada buzzing.

3) Input: footsteps from a hiker.

	Steps	**Input**	**Expected Output**
1	The sensor detects a nearby animal sound.	Sound detected: hiker footsteps	soundData with the raw sound of hiker footsteps.

Integration Test for Feature 1

Feature: Alerting park rangers of the mountain lion detection.

Test Components: Database Class and Animal Audiogram Database API

Test Description: The purpose of this integration test is to verify that the animal sound as a soundData object is properly routed from the Database Class to the API and accurately identified. The components involved in this interaction are animalSounds, storedata(), and generateData() in the Database Class, and animalSounds, isFound, matchedSound, receiveData(), sendClass(), and classifySound() in API Class. The feature must be able to classify the animal sound sent to the API from the Database (in which data is received from the Controller) and send it back to the Database with the correct animal classification. This test aims to cover the following failures: incorrect soundData transferred, misclassification of animal sound, and improper fetching of data from the API.

Test Input:

1) Database.animalSounds = soundData with the raw sound of a mountain lion growl.

Test Expected Output:

1) The database classifies the animal sound as mountain lion:
    - API.receiveData = Database.AnimalSounds
    - API.classifySound() == true
    - API.sendClass() -> “Mountain Lion”
    - Database.generateData() -> “Animal sound: mountain lion”

Integration Testing Steps

1) **Input: soundData with the raw sound of a mountain lion growl.**

	Steps	Input	Expected Output
1	soundData sent to Database from Controller	soundData with the raw sound of a mountain lion growl.	Database.AnimalSounds stores the raw sound.
2	Database.AnimalSounds sent to API	Database.animalSounds	API.receiveData = Database.animalSounds
3	API classifies API.AnimalSounds data	API.animalSounds	API.classifySound() == true
4	Classification sent to Database	API.matchedSound	API.sendClass() -> “Mountain Lion” Database.generateData() -> “Animal sound: Mountain Lion”

<u>System Test for Feature 1</u>

<u>Feature:</u> Alerting park rangers of the mountain lion detection.

<u>Test Components:</u> SensorDetection Class, Controller Class, Database Class, Animal Audiogram Database API.

<u>Test Description:</u> The purpose of this system test is to verify that the major classes properly interact with one another to ensure the total system flow. Start-to-end flow of the system with the major components from detection of mountain lion sounds to alert park rangers. The classes involved in this interaction are SensorDetection, Controller, Database, and API. The feature must detect animal sounds, classify animal sounds, and notify the park rangers (if mountain lions), so they can alert the nearby hikers. The park rangers are not warned of any danger if the animal is not a mountain lion. This test aims to cover the following failures: incorrect soundData transferred throughout the components causing a misclassification, improper fetching of data from the API, and inability of the Controller to to alert the park rangers that a mountain lion is present near the hikers.

Test Input:

1) Mountain lion: growl
2) Dog: bark

Test Expected Output:

1) The park rangers are alerted to the presence of a mountain lion.
2) The park rangers are NOT alerted of any presence of a mountain lion.

System Testing Steps

1) Input: growl from a mountain lion.

	Steps	**Input**	**Expected Output**
1	The sensor detects nearby animal sound	Sound detected: mountain lion growl	soundData with the raw sound of a mountain lion growl.
2	soundData sent to Database from Controller	soundData with the raw sound of a mountain lion growl.	Database.AnimalSounds stores the raw sound.
3	Database.AnimalSounds sent to API	Database.AnimalSound s	API.receiveData = Database.AnimalSounds
4	API classifies API.AnimalSounds data	API.AnimalSounds	API.classifySound() == true
5	Classification sent to Database	API.matchedSound	API.sendClass() -> “Mountain Lion” Database.generateData() -> “Animal sound: Mountain Lion”

6	Controller accesses animal classification	Database.generateData( ) -> “Animal sound: Mountain Lion”	Classification is transferred to Controller via Controller.accessData() Controller.receiveData = “Animal sound: Mountain Lion”
7	The Park ranger is notified of the threat.	Controller.receiveData = “Animal sound: Mountain Lion”	Displays the location of animal sound with a map via Controller.displayData(). Alert the area to alert nearby hikers via Controller.actAlert(). Controller.actAlert() == true.
8	Park ranger turns off the alarm (once hikers are safe, or ML is no longer in the area)	The area is clear. No ML.	Controller.actAlert() == false.

2) **Input: bark from a dog.**

	Steps	**Input**	**Expected Output**
1	The sensor detects nearby animal sound	Sound detected: dog bark	soundData with the raw sound of a dog bark.
2	soundData sent to Database from Controller	soundData with the raw sound of a dog bark.	Database.AnimalSounds stores the raw sound.
3	Database.AnimalSounds sent to API	Database.AnimalSounds	API.receiveData = Database.AnimalSounds
4	API classifies API.AnimalSounds data	API.animalSounds	API.classifySound() == true
5	Classification sent to Database	API.matchedSound	API.sendClass() -> “Dog” Database.generateData() -> “Animal sound: Dog”
6	Controller accesses	Database.generateData()	Classification is transferred to

	animal classification	-> “Animal sound: Dog”	Controller via Controller.accessData() Controller.receiveData = “Animal sound: Dog”
7	The Park ranger is notified of the classification.	Controller.receiveData = “Animal sound: Dog”	Controller.actAlert() == false. The area is clear. No ML.

2.2 Feature 2: Generating a specific report with all relevant and organized information.

Unit Test #1 for Feature 2

Feature: Generating a specific report with all relevant and organized information.

Test Component: DisplayData() in Controller.

Test Description: The purpose of this unit test is to test the controller's ability to be able to display the data such as the location of the detected lion for the verified park ranger. This class displays a User Interface to view and interact with. The failure we are trying to cover for this test case is the wrong data being displayed or the request to display, the next page, and the previous page do not go through.

Test Input: A common user interface option for the verified park ranger to select.

1) Display Data
2) Next page
3) Previous page

Test Expected Output:

1) The Controller displays the data

2) The Controller displays the next page

3) The Controller displays the previous page

Unit Testing Steps

1). Display Data

	Steps	**Input**	**Expected Output**
1	The Controller receives input to display the report	Controller.DisplayData(data)	The Data is displayed

2). Next Page

	Steps	**Input**	**Expected Output**
1	The Controller receives input to display the next page.	Controller.DisplayData(Data.next)	On the next page, data is displayed.

3). Previous Page

	Steps	**Input**	**Expected Output**
1	The Controller receives input to display the previous page.	Controller.DisplayData(Data.previous)	The previous page data is displayed.

Unit Test #2 for Feature 2

Feature: Generating a specific report with all relevant and organized information.

Test Components: Security and Reports classes

<u>Test Description</u>: The purpose of this unit test is to ensure the Security classes' ability to check the access of the system's users. This class will check the user ID of the park ranger who is trying to log into the system. This test aims to prevent users who are not park rangers from generating reports.

<u>Test Input</u>: There are only two possible inputs.

1) Valid Username
2) Invalid Username

<u>Test Expected Output</u>:

1) Allows park rangers to log in and access the system and its functions
2) Displays an error message

<u>Unit Testing Steps:</u>

1) **For valid username:**

	Steps	**Input**	**Expected Output**

1	Enter username and password	ID= CByars
		Password: 12345
		Security.CheckAccess()
			Gives the park ranger access to the user interface and allows the ranger to generate a report

1) **For invalid username:**

	Steps	**Input**	**Expected Output**

1	Enter username and password	ID= CbYarsSS
		Password: 12345
		Security.CheckAccess()
			Displays error message about incorrect username or password

Integration Test #1 for Feature 2

Feature: Generating a specific report with all relevant and organized information.

Test Components: Database Class, Report Class, Controller, Animal Audiogram Database API.

Test Description: The purpose of this integration test is to verify that reports are generated with the correct information per the request of the Controller. The components involved in this interaction are the following Classes: Controller, Database, Report, and API. The feature must successfully transfer the matching information in the database, according to the animal classification from the API and data (location, soundData) from the Controller, to the Report Class to generate a specific report. This report is then sent to the Controller. This test aims to cover the following failures: incorrect and irrelevant information for the report, incorrect report generated, and error in receiving the request to generate a report.

Test Input:

1) The Controller requests a graphical report to read/ view all detections.
   - Controller.genRP("Graphical Report")

Test Expected Output:

1) The Controller accesses a complete graphical report with information about all the detections in the park. This report is in a digital file format that can be printed if needed.

Integration Testing Steps

1) Input: requesting a GraphReport.

	Steps	Input	Expected Output
1	Controller initiates the request.	Controller.genRP("Graphical Report"	Reports.receiveRequest == true Report.reportType =

			“Graphical Report”
2	Database organizes all stored data (soundData, matchedSound, sensorID, sensorLoc.).	Reports.receiveRequest == true	Database.organizeData(animalSounds)
3	Reports accesses organized data and formats it into a GraphReport.	Reports.organizeInfo()	Organized data is compiled into appropriate format with information relevant to Graphical Report.
4	Generate the Graphical Report.	Reports.generateReport()	A completed Graphical Report is generated.

5	Graphical Report is sent to the Controller.	Reports.sendRep()	Controller.receiveData = report
			
			Controller.displayData(report) (as needed)

Integration Test #2 for Feature 2

Feature: Generating a specific report with all relevant and organized information.

Test Component: Controller, data routing, and security.

Test Description: The purpose of this test is to verify that the software system’s security is working properly. This test will involve three classes which are the controller, data routing, and security. The class controller will send the ID and password through the data routing class to reach the security class. The security class will return a boolean value of true or false to represent the status of passing the security test. True means that the ID and Password exist and verify the user. The failure that we are trying to prevent for this test case is the status, “Logged in” does not display.

Test Input: ID and Password

Test Expected Value: The Status "Logged in"

Integration Testing Step:

2) **Input: ID and Password**

	Steps	**Input**	**Expected Output**
1	The controller receives Strings of ID and Password.	ID: akaso234 Password: akeowps23#@ Security.CheckAccess()	Return Void
2	The Controller sends the data through Data Routing	DataRouting.SendData()	Return void
3	The Security Class receives Data.	Security.AcessLog()	Security.GrantAcess == true

4	The security Class sends the data through Data Routing	DataRouting.Receievedata()
		DataRouting.SendData()
			Return void

5	The controller Receives the data.	DataRouting.Receievedata()
		DataRouting.SendData()
			The controller displays, "Logged In"

System Test for Feature 2

Feature: generating and displaying the report.

Test Components: Controller, Security Animal Audiogram Database, Reports, and Database

Test Description: The start-to-end flow of reports being generated. This involves taking the information from the Animal Audiogram database, organizing and generating the data, to finally generating and displaying the report. The failure this test tries to prevent is generating a report for the wrong animal or animals.

Test Inputs:

1) Generate a report of mountain lion detections for the month

2) Generate a report of all non-mountain lion detections for the month

Test Expected Output:

1) A report for all mountain lion detections in the last month is generated

2) A report of all non-mountain lion detections in the last month is generated.

System Testing Steps:

1. **Generate report for mountain lion detections**

	Steps	**Input**	**Expected Output**
1	Park Ranger enters login information	ID: CByars password: 12345 Security.CheckAccess()	Security.checkAccess == true
2	Park Ranger navigates user interface to generate mountain lion report	Ranger goes through the controller and clicks button to generate report	The system begins the process of generating a report
3	The Animal Audiogram Database categorizes animal noises	API.classifySound() ->Animal Sound: Mountain Lion	Takes all detections made by mountain lions and begins classifying them accordingly
4	receiveRequest() is called to see if a report has been requested	Reports.receiveRequest	Reports.receiveRequest == true
5	Once the system sees that receiveRequest is true, generateData() takes animal sounds and only grabs the	Database.generateData()--> "Animal Sound: Mountain Lion"	Reports.generateReport = Database.generateData()

	Steps	Input	Expected Output
1	Park Ranger enters login information	ID: CByars password: 12345 Security.CheckAccess()	Security.checkAccess == true
2	Park Ranger navigates user interface to generate mountain lion report	Ranger goes through the controller and clicks button to generate report	The system begins the process of generating a report
	mountain lion sounds		
6	The controllers receives and displays the generated report	Reports.sendRep()	Controller.DisplayData()= Reports.sendRep()

2. **Generate report for non mountain lion detections.**

	Steps	Input	Expected Output
1	Park Ranger enters login information	ID:CByars password: 12345 Security.CheckAccess()	Security.checkAccess == true
2	Park Ranger navigates user interface to generate a report for all animals that are not mountain lions	Ranger goes through the controller and clicks button to generate report	The system begins accessing the animal audiogram database
3	The Animal Audiogram Database categorizes animal noises	API.classifySound() -> Animal Sound: All non mountain lion sounds	Takes the detections of all animals except mountain lions and begins classifying them accordingly

	Steps	Input	Expected Output
1	Park Ranger enters login information	ID:CByars password: 12345 Security.CheckAccess()	Security.checkAccess == true
2	Park Ranger navigates user interface to generate a report for all animals that are not mountain lions	Ranger goes through the controller and clicks button to generate report	The system begins accessing the animal audiogram database
4	receiveRequest() is called to see if a report has been requested	Reports.receiveRequest	Reports.receiveRequest == true
5	Once the system see's that receiveRequest is true, generateData() takes animal sounds and only grabs the mountain lion sounds	Database.generateData()--> "Animal Sound: Mountain Lion"	Reports.generateReport = Database.generateData()
6	The controller receives and displays the generated report	Reports.sendRep()	Controller.DisplayData()= Reports.sendRep()

3. Team Members' Responsibilities

<u>Christian Byars</u>

-Grammar, format, and capitalization

-Feature 2: Unit Test #2

-Feature 2: System Test #2

<u>Nuonnettra Kanzaki</u>

Feature 2: Unit Test #1

Feature 2: Integration Test #2

<u>Kyle Camposano</u>

- Responsible for completing 4 test cases:
  - Feature 1: Unit Test
  - Feature 1: Integration Test
  - Feature 1: System Test
  - Feature 2: Integration Test
- Responsible for drafting a brief overview of the Test Plan for Verification.
- Responsible for drafting and detailing section 4 (Updates on the Software Design Specification)

4. Updates on the Software Design Specification

This section describes the changes made to the original version of the Software Design Specification. The changes were minimal, yet necessary to ensure clarity and flow for the test plan phase.

<u>UML Class Diagram (SDS 2.3)</u>: Reports Class now shares a dependency with Database Class. This change is necessary as the Reports Class needs to directly access the stored data (classifications, location, and animal sounds) in the Database Class. Without this change, there's no bridge to the gap between the Database and Reports Classes in the test cases.

Database Class (SDS 2.4.2): The description of Database Class now matches the change on the UML Class Diagram. The Database Class specifically mentions the interaction (dependency) of the Reports Class to the Database Class.