Software Design Specification: Mountain Lion Detection System

Prepared by Christian Byars, Nuonnettra Kanzaki, and Kyle Camposano

11th October 2024

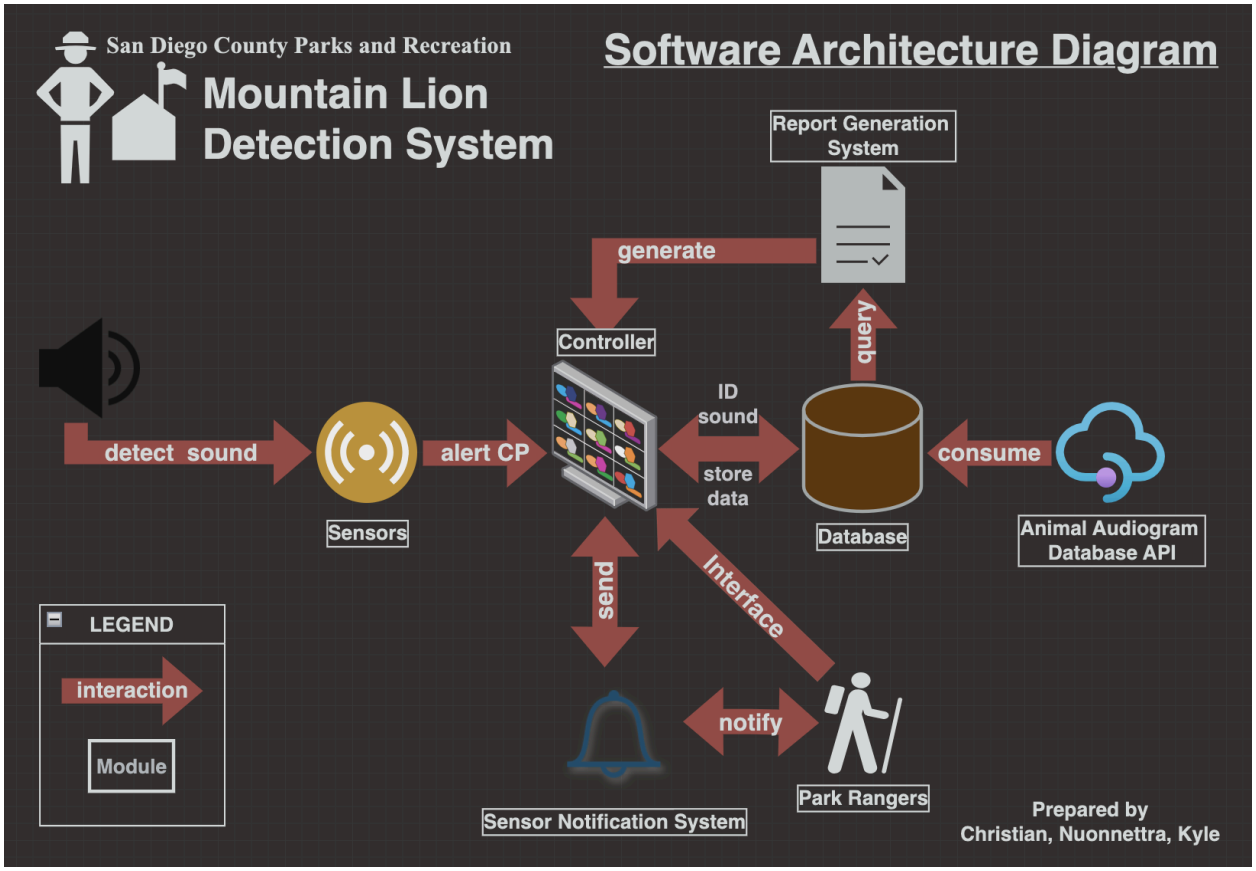
1. System Description

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



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.

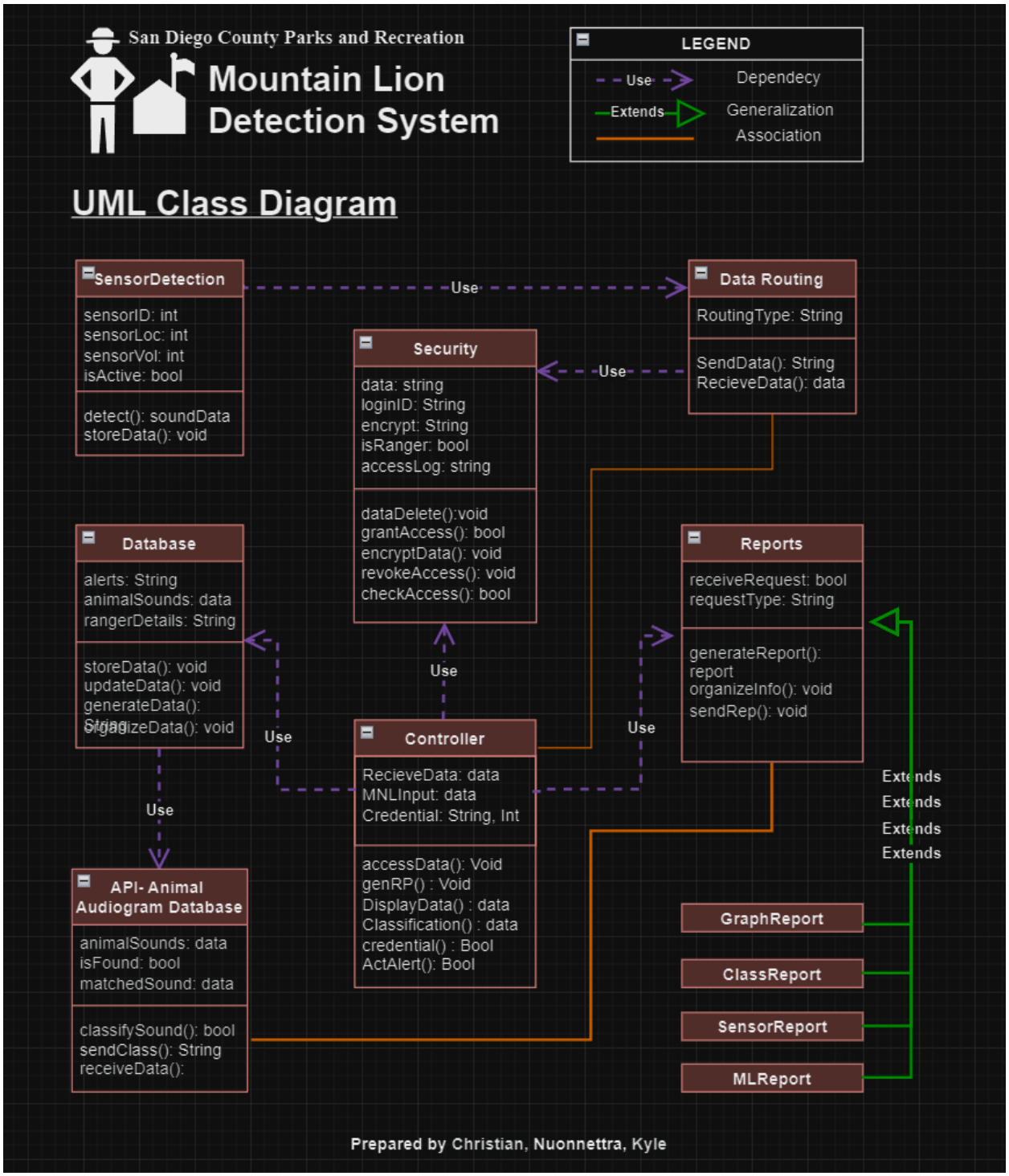
5. **Module:** Controller
    a. **Description:** The controller is the most crucial component of the software architecture. It is where most of the information and interaction go. This is where the data about the possible mountain lion will be sent, which in turn, the control program will send to the 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 and the 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.

Attributes

- 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.

Operations

- 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) **Class**

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.

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(): give the user the option to visually view the data including a map. 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.
- Proofcheck 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