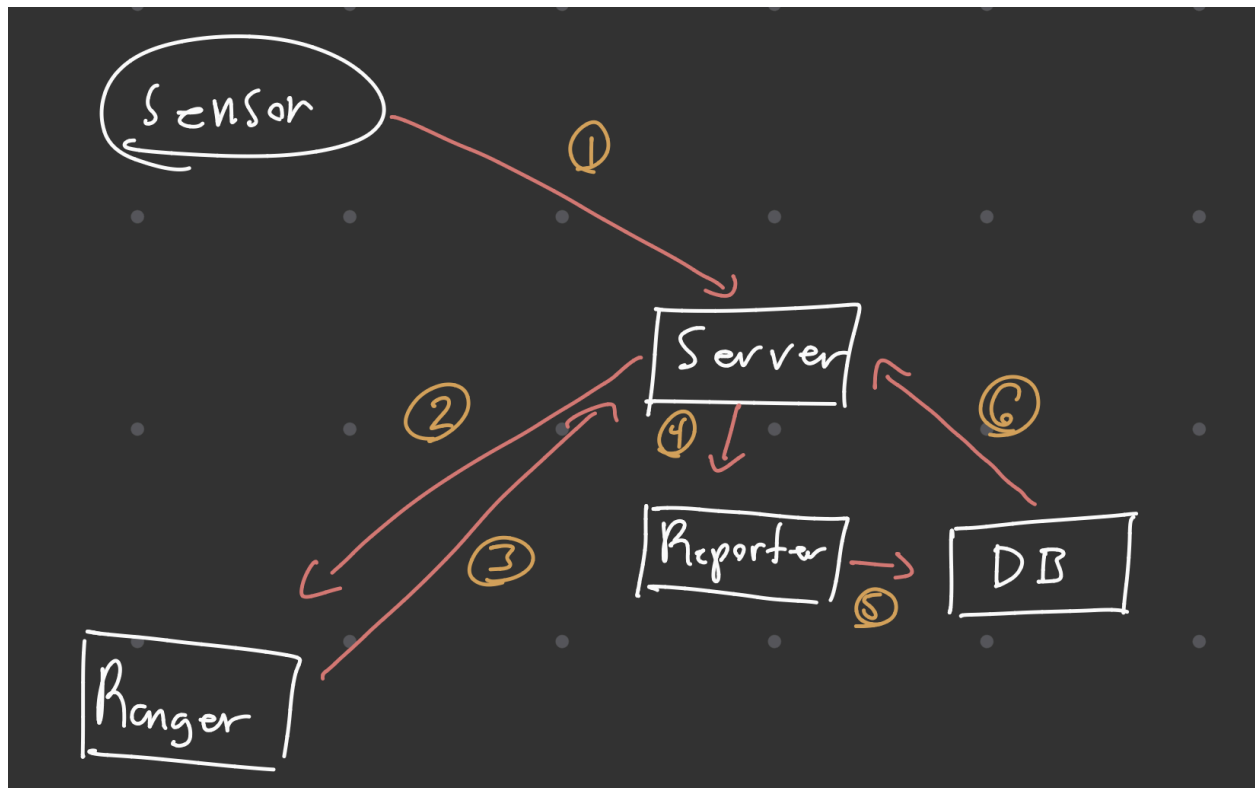


# Mountain Lion Detection System

Names: Ethan Dinehart, Ryan Thleiji, Rohan Sen, Michael Reese

**System Description:** The mountain lion detection system is a system of noise detection devices that San Diego County Parks and Recreations Department will use to track, record, and be alerted Rangers of mountain lions in a 5 square mile area chosen for deployment. The system will be able to detect sound, and alert rangers on animal type, signal strength, and location of detection, store the alert for 30 days and summarize data for up to a year. The system will also allow rangers to report alerts, report/generate a visual graphical analysis, and report animal classifications.

## Software Architecture Overview



## Description:

The alert from the sensor goes to the server(controlling Computer). This then sends the alert to Ranger's device, once he classifies it that data is sent back to the server. From here the data is all sent to the report generator that creates the report then stores it in a database. Whenever the

ranger's device requests a report it's sent to the server then pulled from the database and sent back to the server then the ranger.

Architectural diagram of all major components

UML Class Diagram

<https://drive.google.com/file/d/1oJOYPWM0BXkTR8KzN-vG-54ceySISd33/view?usp=sharing>

Description of classes

Description of attributes

Description of operations

\* descriptions should be detailed and specify datatypes, function interfaces, parameters, etc..

### **Sensor**

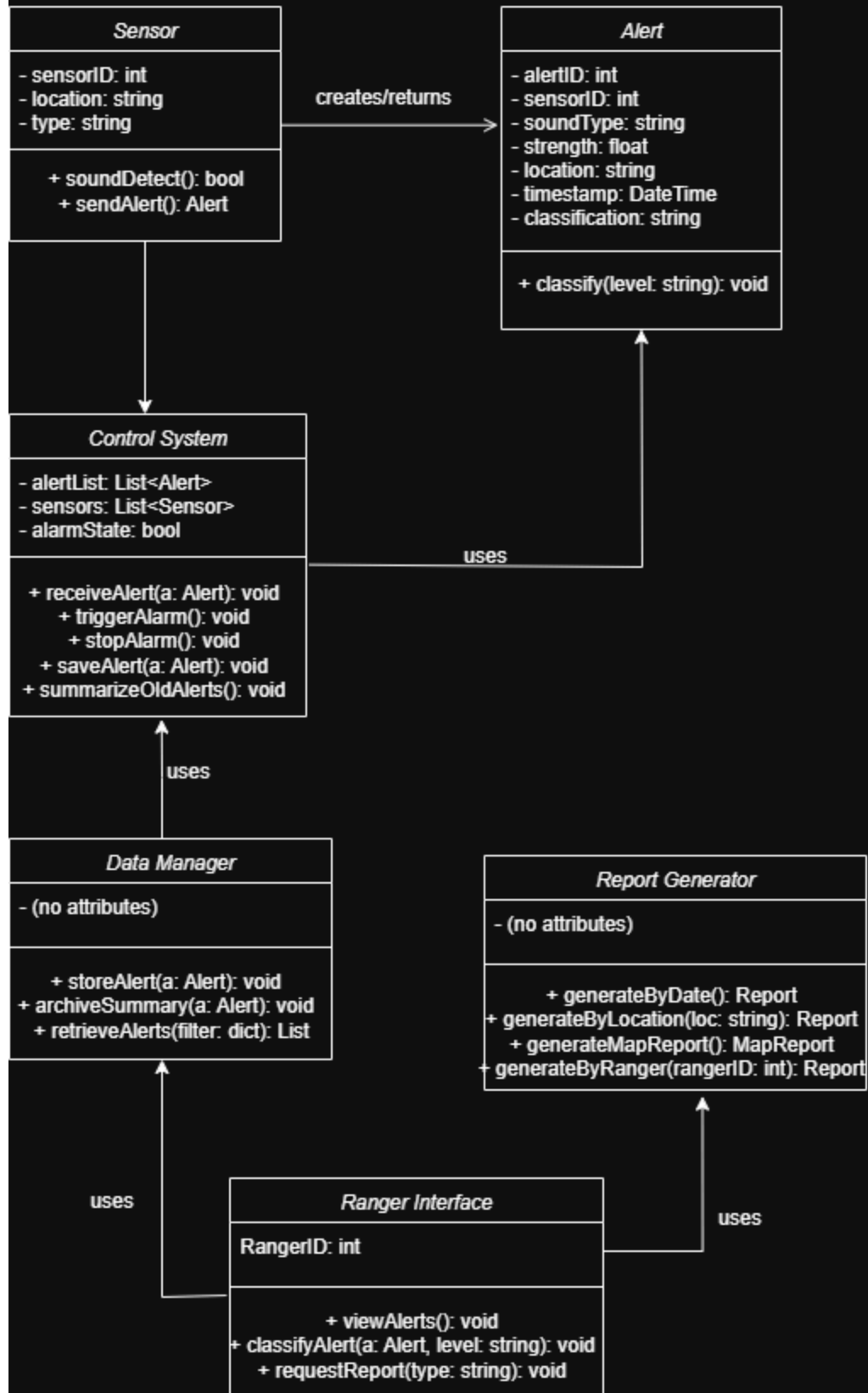
- **Attributes:**

- sensorID: int – Unique sensor identifier (help differentiate sensors).
- location: string – Location for the sensor (gps).
- type: string – Type of animal sound sensed

- **Operations:**

- soundDetect(): bool – Activates when target sound detected.
- sendAlert(): Alert – Sends alert to control system.

# UML Diagram



Description: The UML Diagram shows the structure of the MLDS and how its parts interact. The sensor class detects animal noises and creates an alert, which is made up of details like type, strength, location, date, etc. The control system receives the alert and communicates with the data manager to store the data. The report generator produces reports based on alerts and ranger classifications. The ranger interface allows rangers to view and classify the reports.

## **Alert**

- **Attributes:**

- alertID: int - Alert classification
- sensorID: int - the sensorID that triggered the alert
- soundType: string - animal sound type that was detected by the sensor
- strength: float - strength of the sound type
- location: string - location where the alert was triggered
- timestamp: DateTime - time when the alert was triggered
- classification: string - animal classification by the alert

- **Operations:**

- classify(string): void – Sets classification (definite, suspected, false).

## **ControlSystem**

- **Attributes:**

- alertList: List<Alert>
- sensors: List<Sensor>
- alarmState: bool

- **Operations:**

- receiveAlert(Alert): void - Handles incoming alerts (rangers can choose to not get an alarm if animal sound detected isn't a mountain lion)
- triggerAlarm(): void - Activates alarm system when alert is received
- stopAlarm(): void - Deactivates alarm system once rangers turns off

- saveAlert(Alert): void - sends alert in database
- summarizeOldAlerts(): void - Archive and summarize alerts

### **DataManager**

- Handles data persistence (database or local storage).
- **Attributes:**
  - None
- **Operations:**
  - storeAlert(Alert): void
  - archiveSummary(Alert): void
  - retrieveAlerts(dict): List<Alert>

### **ReportGenerator**

- **Operations:**
  - generateByDate(int) - generate a report by date
  - generateByLocation(loc) - generate a report by location
  - generateMapReport() - generate a report of the map of alerts
  - generateMountainLionReport() - generates a report of only mountain lions.
  - generateClassification() - generates a report off rangers animal classification
  - generateByRanger(rangerID) - generate a specified report per rangers requests

### **RangerInterface**

- Provides GUI for rangers
- **Attributes:**
  - RangerID: int - unique ranger ID for specified ranger
- **Operations:**
  - viewAlerts() - allows the rangers to view alerts

- `classifyAlert(level,rangerID)` - allows the ranger to classify the alert as (definite, suspected, or false)
- `requestReport(type)` - allows the rangers to request reports to view a graphical analysis of data

## Development Plan & Timeline

### Overview of the Software System

The mountain lion detection system will help park rangers in San Diego County Parks identify and respond to possible mountain lion activity.

### Partitioning Tasks

Task	Description
<b>System Design</b>	Define how sensors communicate with the control computer, database setup, and alarm logic.
<b>Database Development</b>	Create tables to store alerts, classifications, and ranger information. Set up 30-day and 1-year data storage.
<b>Alert Communication Module</b>	Handle incoming sensor data (type, strength, location). Trigger alarm when alerts are received.
<b>User Interface (UI) Development</b>	Build screens for viewing alerts, stopping alarms, classifying alerts, and generating reports.
<b>Report Generation</b>	Create reports by date, by sensor, on a map, and by ranger.
<b>Testing and Debugging</b>	Test everything, fix issues, and verify data accuracy.
<b>Deployment</b>	Install the system at the ranger station.

### Timeline

**Week 1-2:** Project planning, system design.

**Week 3-4:** Database creation, alert communication module.

**Week 5:** UI development

**Week 6-7:** Reporting features and classification system

**Week 8:** Testing and fixing bugs

**Week 9:** Final product

### **Team Member Responsibilities**

<b>Team Member</b>	<b>Responsibilities</b>
<b>Project Manager</b>	Oversees schedule and coordinates team communication, ensures the project is completed on time.
<b>System Designer</b>	Creates system architecture and ensures compatibility with sensors.
<b>Database Developer</b>	Designs and implements the database(s).
<b>Software Developer</b>	Builds alert handling logic, alarm system, and classification functions.
<b>UI Developer</b>	Designs user interface for alerts, reports, and maps.
<b>Tester(s)</b>	Tests all parts of code (many possible test cases), verifies requirements are met, and reports bugs.