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

Architectural diagram of all major components
UML Class Diagram
https://drive.google.com/file/d/1oJOYPWM0BXkTR8KzN-vG-54ceySISd33/view?usp=sharing

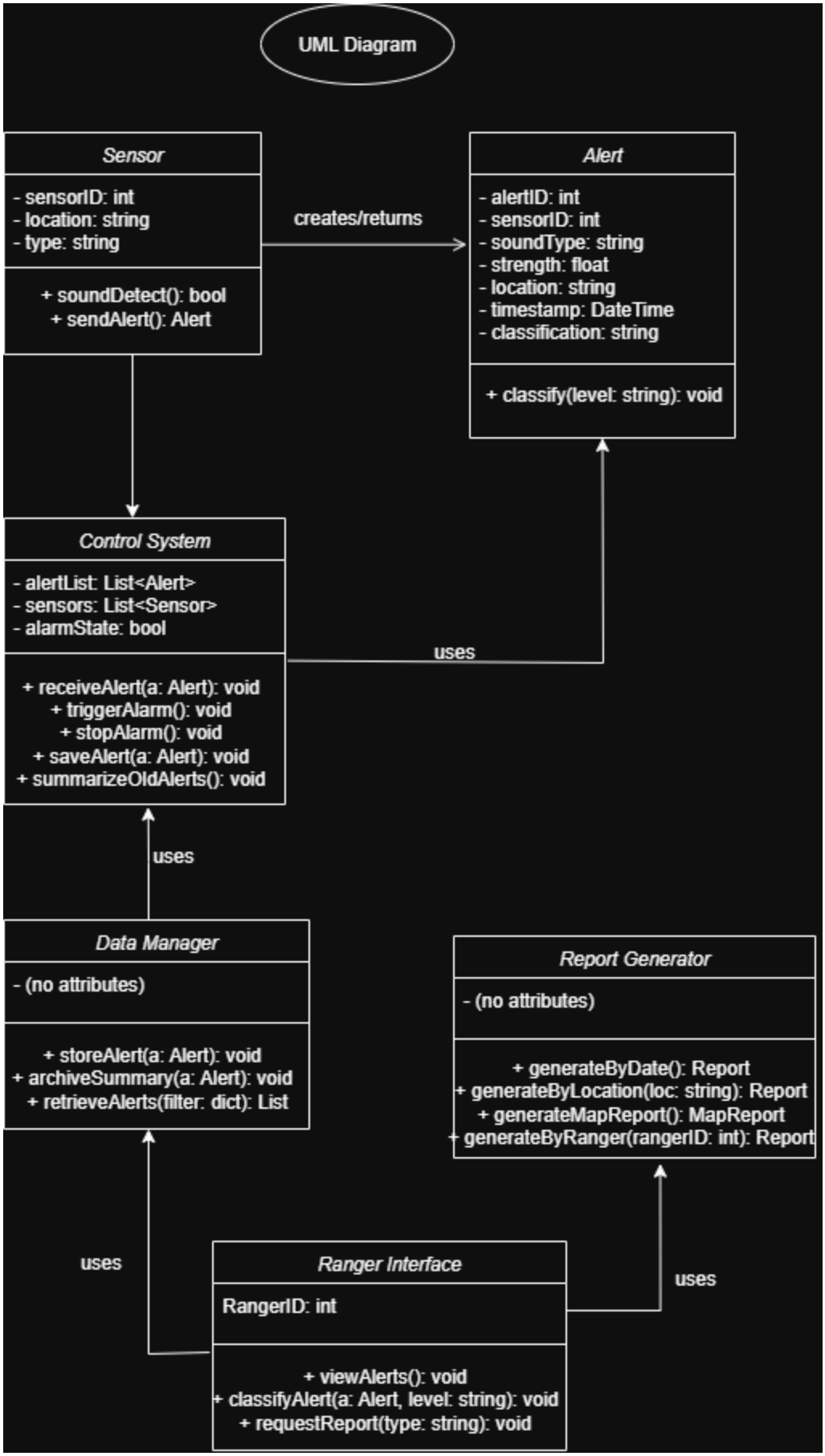
Description of classes
Description of attributes
Description of operations

* 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
Sensor
- sensorID: int
- location: string
- type: string
+ soundDetect(): bool
+ sendAlert(): Alert
creates/returns
Alert
- alertID: int
- sensorID: int
- soundType: string
- strength: float
- location: string
- timestamp: DateTime
- classification: string
+ classify(level: string): void
Control System
- alertList: List<Alert>
- sensors: List<Sensor>
- alarmState: bool
+ receiveAlert(a: Alert): void
+ triggerAlarm(): void
+ stopAlarm(): void
+ saveAlert(a: Alert): void
+ summarizeOldAlerts(): void
uses
uses
Data Manager
- (no attributes)
+ storeAlert(a: Alert): void
+ archiveSummary(a: Alert): void
+ retrieveAlerts(filter: dict): List
Report Generator
- (no attributes)
+ generateByDate(): Report
+ generateByLocation(loc: string): Report
+ generateMapReport(): MapReport
+ generateByRanger(rangerID: int): Report
uses
Ranger Interface
RangerID: int
+ viewAlerts(): void
+ classifyAlert(a: Alert, level: string): void
+ requestReport(type: string): void
uses

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

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