Mountain Lion Detection System Software Design Specification

Created by Vince Alihan, Carson Mucho, and Kaelin Facun

I. System Description:

This software system was requested by the San Diego Parks and Recreation. The purpose of this software is to detect mountain lions in various state parks throughout San Diego County. The San Diego Parks and Recreation wants this system to be able to warn patrons who enter the parks, that if a mountain lion is nearby, a park ranger will tell them to move away from the area as it is unsafe. The idea is that there will be various sensors throughout the park. The system will use an animal detection system created by the Animals-R-Here company to ping different locations of various animals and update the park's database. The system is designed to only be used by park rangers to see where animals are for themselves, so only they can access the data. This document will break down the various requirements such as user, system (functional and non-functional), and other features that this software system is specified to include.

II. Software Architecture Overview:

Below is our designated software architecture diagram, as well as our UML diagram for the Animals-R-Here system. Given the California State Parks system's interest in adopting our program, our detection system heavily depends on servers and databases accessible through a wifi connection and a computer.

Animal-R-Here Architecture

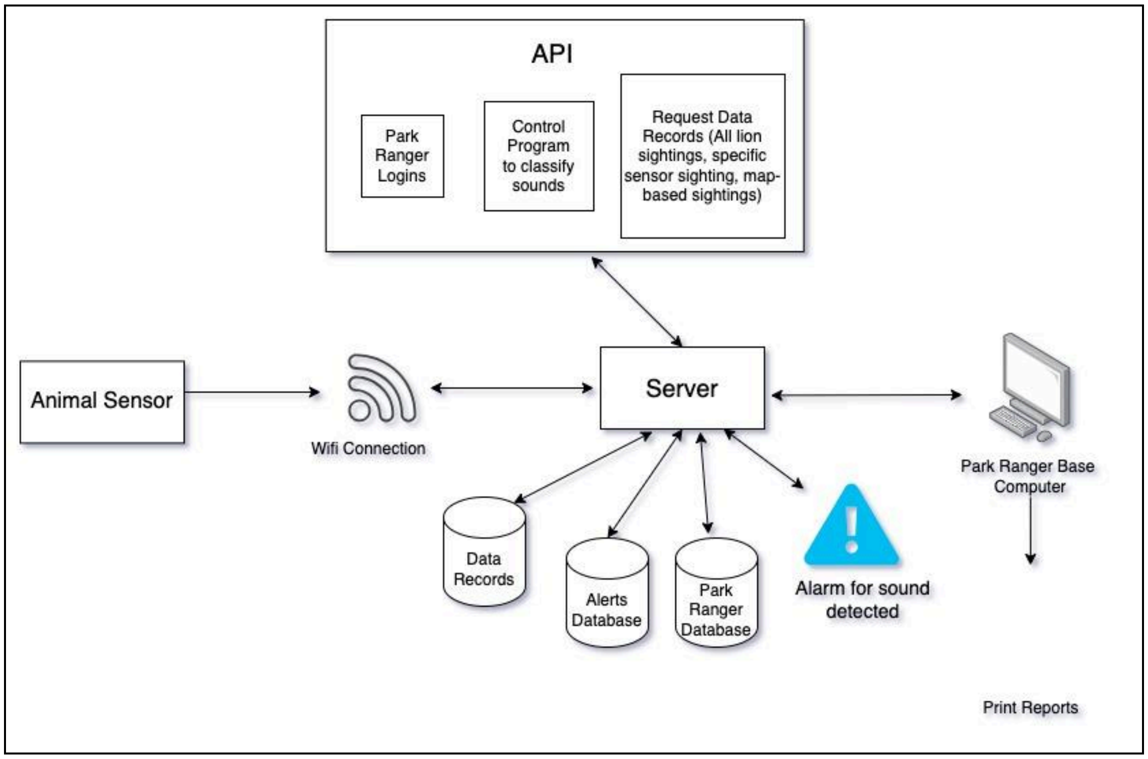
API
Park Ranger Logins
Control Program to classify sounds
Request Data Records (All lion sightings, specific sensor sighting, map-based sightings)
Animal Sensor
Wifi Connection
Server
Park Ranger Base Computer
Data Records
Alerts Database
Park Ranger Database
Alarm for sound detected
Print Reports

Figure 1.1: Animal-R-Here Software Architecture

Our software system begins with the **Animal Sensors** to detect mountain lions in close proximity, which are connected to **Wifi** in order to relay information to the **Server**. The **Server** is connected to 3 Databases, such as the **Data Records** which contains all mountain lion detections by date and all the locations they were detected at and were already classified by the ranger, the **Alerts Database**, which contains all alerts detected which are for the Park Ranger to classify, and the **Park Ranger Database**, which contains all Park Ranger information such as user preferences and information specific to the ranger logged in. These databases send and receive information to and from the Server. The Server is also connected to an **Alarm**, which sounds whenever an alert message is received from the animal detection system. Attached to the Server is the **Application Programming Interface (API)**, which provides functionality for the server information, to be used by the **Park Ranger Base Computer** connected to the server. In this interface, functionality is provided to allow park rangers to log into the computer, a control program which allows the park ranger to classify sounds detected as definite, suspected, or false, and an option to request data records stored in the database. All of these functions can be accessed by the Park Ranger Base Computer connected to the Server, which can print reports with the information provided by the server connected to the databases, such as all mountain lion detections by date, all detections at a specific location, a graphical report showing detections on a map, and a report showing detection classifications by ranger.

Animal-R-Here UML Diagram

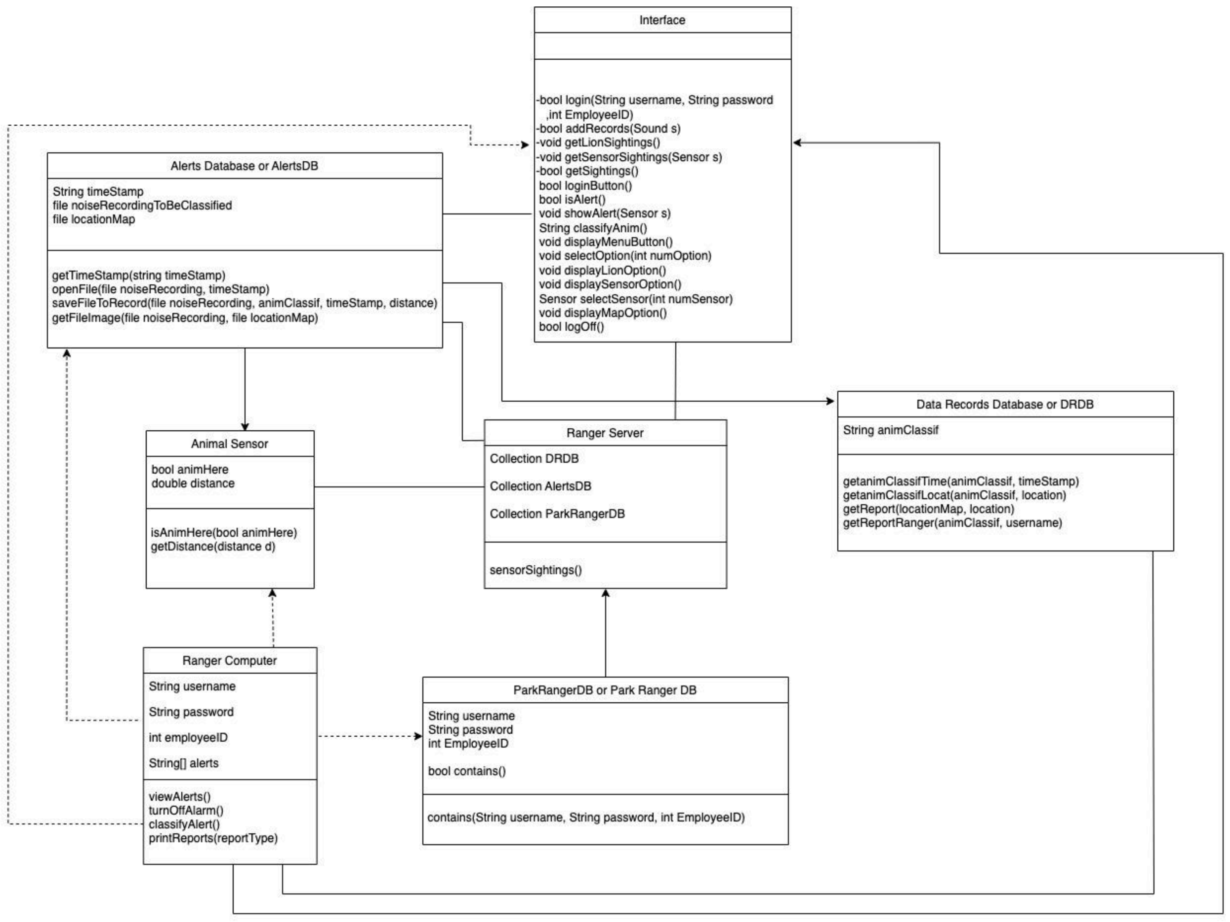
Interface
-bool login(String username, String password ,int EmployeeID)
-bool addRecords(Sound s)
-void getLionSightings()
-void getSensorSightings(Sensor s)
-bool getSightings()
bool loginButton()
bool isAlert()
void showAlert(Sensor s)
String classifyAnim()
void displayMenuButton()
void selectOption(int numOption)
void displayLionOption()
void displaySensorOption()
Sensor selectSensor(int numSensor)
void displayMapOption()
bool logOff()
Alerts Database or AlertsDB
String timeStamp
file noiseRecordingToBeClassified
file locationMap
getTimeStamp(string timeStamp)
openFile(file noiseRecording, timeStamp)
saveFileToRecord(file noiseRecording, animClassif, timeStamp, distance)
getFileImage(file noiseRecording, file locationMap)
Data Records Database or DRDB
String animClassif
getanimClassifTime(animClassif, timeStamp)
getanimClassifLocat(animClassif, location)
getReport(locationMap, location)
getReportRanger(animClassif, username)
Animal Sensor
bool animHere
double distance
isAnimHere(bool animHere)
getDistance(distance d)
Ranger Server
Collection DRDB
Collection AlertsDB
Collection ParkRangerDB
sensorSightings()
Ranger Computer
String username
String password
int employeeID
String[] alerts
viewAlerts()
turnOffAlarm()
classifyAlert()
printReports(reportType)
ParkRangerDB or Park Ranger DB
String username
String password
int EmployeeID
bool contains()
contains(String username, String password, int EmployeeID)

Figure 1.2: Animal-R-Here UML Diagram

Animal Sensors: The animal sensor is connected to the Alerts Database from the Ranger Server as well as the Ranger Computer. The sensor program contains *bool animHere and double distance*.

- *bool isAnimHere(bool animHere)*: This is the alert that sets off the indication that an animal is in the vicinity of the sensor. Once indicated, the alarm sets off the sound recording and takes track of the time alerted. In the Ranger computer, the command *turnOffAlarm()* turns this boolean to false to turn off the alarm until there is another sound in the vicinity.

- *getDistance(distance d)*: This function collects the distance of the sound and tracks the vicinity from the sensor. This distance is collected and recorded in the databases to be used from the Alert and Data Record databases.

Ranger Computer: The ranger computer is the main piece of hardware used to view and control every aspect of the software system. The ranger computer takes in a string input of a username and password, and an int input of the user's employee ID in order to verify that the user is authorized to access the system. There is also an array of strings named alerts to store any alerts the animal sensor sends to the computer for the ranger to view.

- *viewAlerts()*: This is one of the main functions a ranger can perform from their computer, which is to view any alerts that the animal sensor sends to the Alerts Database, which the computer can access
- *turnOffAlarm()*: When movement is detected by the animal sensors, it sends an alarm to the computer alerting that movement is detected, so with this function, a ranger can shut off this alarm to indicate after it has been acknowledged.
- *classifyAlert()*: When an alarm is detected, the ranger can view the alert and analyze the sound captured, and decide whether the sound is a definite mountain lion spotting, a suspected spotting, or a false alarm. After this, it is stored into the Alerts Database.
- *printReports(reportType)*: A ranger can choose to print out reports based on the parameter they select which could be about all mountain lion detections by date and classification, a report about all detections at a specific sensor location, report showing detections on a map of the park and areas within 2 miles of the park, and a report showing detection classifications by ranger.

Ranger Server: The Ranger Server makes up the entirety of the system by keeping the collections of the databases of the Data Records database, Alerts database, and the Park Ranger database. The server contains *sensorSightings()*.

- *sensorSightings()*: Retrieves a report from all three databases to print out a specified report requested by the Park Ranger. Prints out the location, animal classification, Park Ranger who classified it, and the graphical location of the map.

Interface: The interface of the software controls what the user (park rangers) are able to do with the software system and is what the user sees when accessing the computer. Currently, there are several methods that the user can use and access from the interface. The following methods are included in the Interface: login(String username, String password, int EmployeeID), addRecords(Sound s), getLionSightings(), getSensorSightings(Sensor s), getSightings(), loginButton(), isAlert(), showAlert(Sensor s), classifyAnim(), displayMenuButton(), selectOption(int numOption), displayLionOption(), displaySensorOption(), selectSensor(int numSensor), displayMapOption(), logOff()

- *login(String username, String password, int EmployeeID)*: this method will verify that the user who is trying to login is a park ranger. By using the parameters of username, password, and EmployeeID, it searches for that criteria in the ParkRangerDB to verify the user is allowed to access the Sighting information. Will have a boolean return type; returns true if the user has access and will then process, otherwise will return false and kick the user out.
- *addRecords(Sound s)*: the addRecords method will allow the user to add the new dataRecord to the AlertsDB. The method will take the alert parameter to add for when new reports are requested/ The alert includes information like animal

Classification, timeStamp, and distance from sensor. Will have a boolean return type, and will return true every time since the user can always add data

- get*LionSightings()*: this method will retrieve all of the confirmed Lion Sightings based on the park rangers confidence of the sound. It will iterate through all of the alerts in the database and retrieve all of the ones that are positive for lion sightings. The method then returns a list of Alert type sightings to the user
- *getSensorSightings(Sensor s)*: This method gets all of the alerts from a specific sensor. It takes in the Sensor parameter and then returns a List of all the sightings from that sensor to the user.
- *getMapOfSightings()*: This method will retrieve all Sightings within a 2 mile radius of the park. It iterates through the alert database and uses the sensor number and sighting distance to populate a list that has all of those sightings. This list is then returned to the user.
- *loginButton()*: This method enacts once the user has put their information in the designated boxes. Inside it uses the login method from the Interface class using the user imputed data. Returns a bool that is based on the return of login() from Interface.
- *isAlert()*: This method is automatically checked once the user logs in. This method checks all of the sensors to see if there was an alert. If there was, it updates a variable with which the sensor was alerted and returns true. If no detection, then returns false.
- *showAlert(Sensor s)*: Ran once isAlert is checked and confirms there was a detection. Takes in the parameter of a Sensor which is pulled from when isAlert()

is run. Once the Sensor has been confirmed it will display on screen the sound and location and the user then needs to classify it

- *classifyAnim()*: Once the user has listened to the sound of the animal they are given three options; definitely a mountain lion, could be a mountain lion, and not a mountain lion. Once a choice has been made returns a String of the classification to be added to the records.
- *displayMenuButton()*: Once the user has been verified through the loginButton() method and an alert has been taken care of the displayMenuButton is automatically run. The Menu consists of 4 possible choices which are displaying all of the mountain lion sightings, displaying every detection from a specific sensor, and displaying a visual representation of all the sightings within two miles of the park.
- *selectOption(int numOption)*: Each option is given a number and based on what the user clicks on, updates the numOption variable which is the parameter of this method. Using that number makes sure that the right option is run and will then run that method.
- *displayLionOption()*: If the user selects option one, they will be shown a list of all confirmed mountain lion detections. This is done by the database being iterated and sorting by animal classification.
- *selectSensor(int numSensor)* This helper method is inside displaySensorOption() which helps the user determine which sensor they want to show from. It will gather every detection from the sensor number put into the parameter and return a list of alerts.

  - *displaySensorOption()*: When this second option is displayed, the user specifies which sensor they would like to see results from and then using the helper method selectSensor() displays those results.
  - *displayMapOption()*: The third and final data option will display a map of sightings that are within two miles of the park. The alert database is iterated through and every detection within two miles is added to a list and is then displayed on screen.
  - *logOff()*: The last option the user can pick is to log off the system. This resets the program back to the start, before the user logged in. Returns true if done successfully, which should be every time when it is run.

Databases: All of the databases of our software system are located in our servers. Currently, we have four databases dedicated to the San Diego County Parks and Recreation Department, but in the future, we plan on building multiple servers for multiple locations across California, if California State Parks would like to expand out of San Diego.

- **Alerts:** The Alerts Database contains all of the new detected recordings that need to be classified by the Rangers under the detected, suspected, and false labels. The Database contains *String timeStamp*, *file noiseRecordingToBeClassified*, as well as *file locationMap.*
  - *String timeStamp:* Holds the time when the alert was made. Can be reached through the Ranger Computer through *printReports(parameter[reportType])*.
    - *getTimeStamp(string timeStamp)*: Retrieves the timestamp of when the recording was recorded.

    - *openFile(file noiseRecordingToBeClassified, timeStamp)*: *openFile()* opens the file with the *file noiseRecordingToBeClassified* as well as *timeStamp*, to associate the noise file and the time recorded.
    - *saveFileToRecord(file noiseRecording, timeStamp)*: Saves the file to Data Records Database to be kept for future reference for retrieval and information.
    - *getFileImage(file noiseRecording, file locationMap)*: Retrieves the alert's graphical map image location as well with the noise recording.
  - *file noiseRecordingToBeClassified*: Holds the noise recording in a file within the database that needs to be classified by the Park Ranger.
- **Park Ranger:** The Park Ranger Database consists of all of current Park Rangers within the San Diego Parks and Recreation Department and holds their account information for logins. Park Ranger holds *String username* and *String password*, as well as *int EmployeeID*. There is also *bool contains()* to check if the account is active and valid through our database.
  - *contains()*: *contains()* as mentioned above, checks if the account is valid and holds the parameters of *contains(String username, String password, int EmployeeID)*.
- **Data Records:** The Data Records Database contains the classified file records of all of the alerts that have been previously analyzed by a park ranger. The database contains *String animClassif* that holds the value of the classification given by the Park Ranger.
  - *String getanimClassif*: Holds the value of the classification that has been analyzed by the Park Ranger.

- *getanimClassifTime(animClassif, timeStamp)*: Retrieves the animal classification and the time of the recording of the alert.
- *getanimClassifLocat(animClassif, location)*: Retrieves the animal classification and the distance of the alert from the sensor.
- *getReport(locationMap, location)*: Retrieves the file of the graphical image from the Alerts Database and the distance from the sensor.
- *getReportRanger(animClassif, username)*: Retrieves the animal classification and the Park Ranger who made the classification report.

III. Development Plan and Timeline:

The current team members of the Animal-R-Here consist of three team members: Carson Mucho, Vince Alihan, and Kaelin Facun.

Carson Mucho is our Lead Back-End Developer and is responsible for the back-end development and system of our project. As a Lead Back-End Developer, he is responsible for the system's operating systems and interactions with the databases and servers, and must ensure that the system works properly back-wise.

Vince Alihan is our Lead Database Architect and is responsible for the databases required for the Animal-R-Here Project. As a Database Developer, he is responsible for the prompt and efficient organization of information that will be provided by local park rangers, as well as upkeep the system based on the user's needs as the project progresses outside of California.

Kaelin Facun is our Lead Front-End Developer and is responsible for the front-end development of the interface of our project and is responsible for ensuring the project fulfills our park ranger's requirements in a system. As a front-end developer, she is responsible for the user interface and their interactions with the system and must ensure that it works properly front-wise.

Animal-R-Here Project will be expected to be released before the Spring 2025 or March 2025. As hiking trails and park visitor occupancy peaks during spring season, we want to make sure that the Animal-R-Here program is installed in local parks to ensure public safety of both visitors and mountain lions alike.

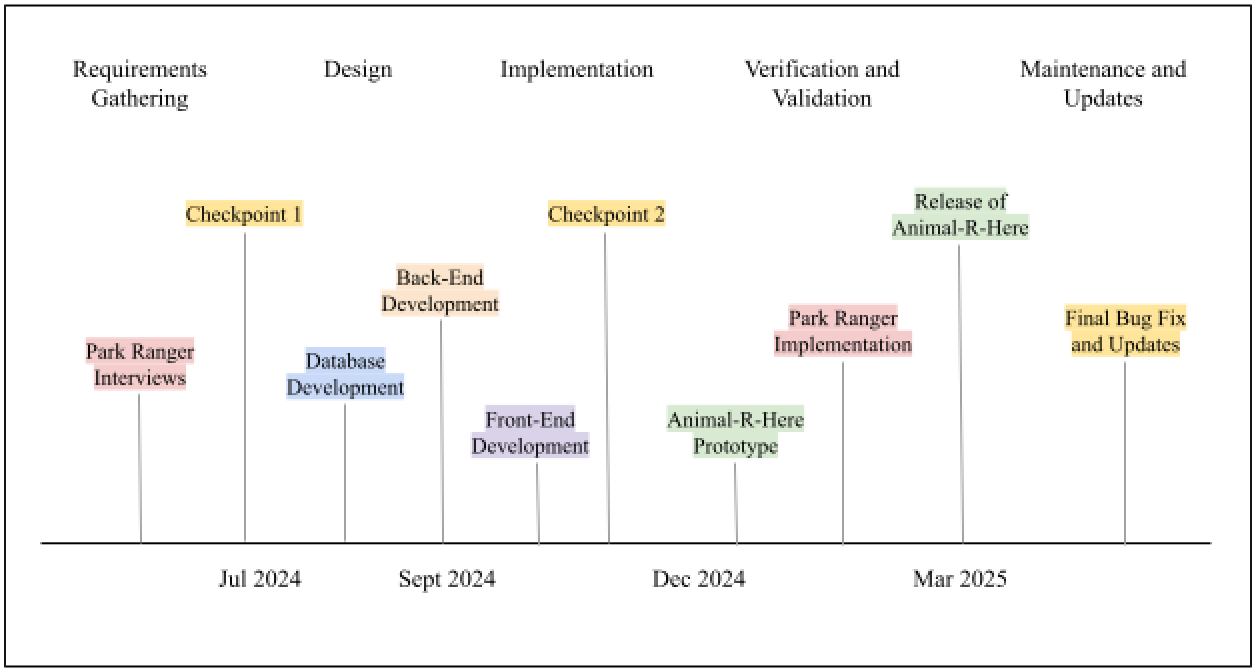

Figure 1.3: Expected Timeline for the Animal-R-Here Project.

Once the project is approved in March, 2024, the Animal-R-Here team can be expected to conduct Park Ranger Interviews as the team can gather requirements of what the park rangers would like to see through this project. Through these interviews, the project team can clarify expectations as well as set realistic expectations for the desired end goal.

Checkpoint 1 and Checkpoint 2 are used for meetings within the team and the client to ensure that the expectations are being met with the project, i.e as interface appearance and functions and database functionality.

From July 2024 and beginning of Dec 2024, the project team is expected to start development on the Animal-R-Here project. Vince Alihan, will start developing the databases and collections and work alongside Carson Mucho to ensure storage and project functionality from database to back-end development. Kaelin Facun will also start on development with the front interface and work alongside Mucho as well to ensure functionality with back-end and front-end development.

The team is expected to release a prototype beta at the beginning of Jan. 2024. Once the Animal-R-Here Prototype is released, the San Diego Parks and Recreation Department will test the prototype at a small, secure location in San Diego to ensure satisfaction with the project and that it fulfills their needs as Park Rangers. The timeline also provides space from testing if the project is still not up to par or for bug fixes in the original code.

Once the project is satisfactorily completed, the final release of Animal-R-Here will happen in Mar. 2024, and further maintenance and updates will be conducted after the release to ensure any further bug fixes as well as updates with the program.