Kilimanjaro Trail Monitor System

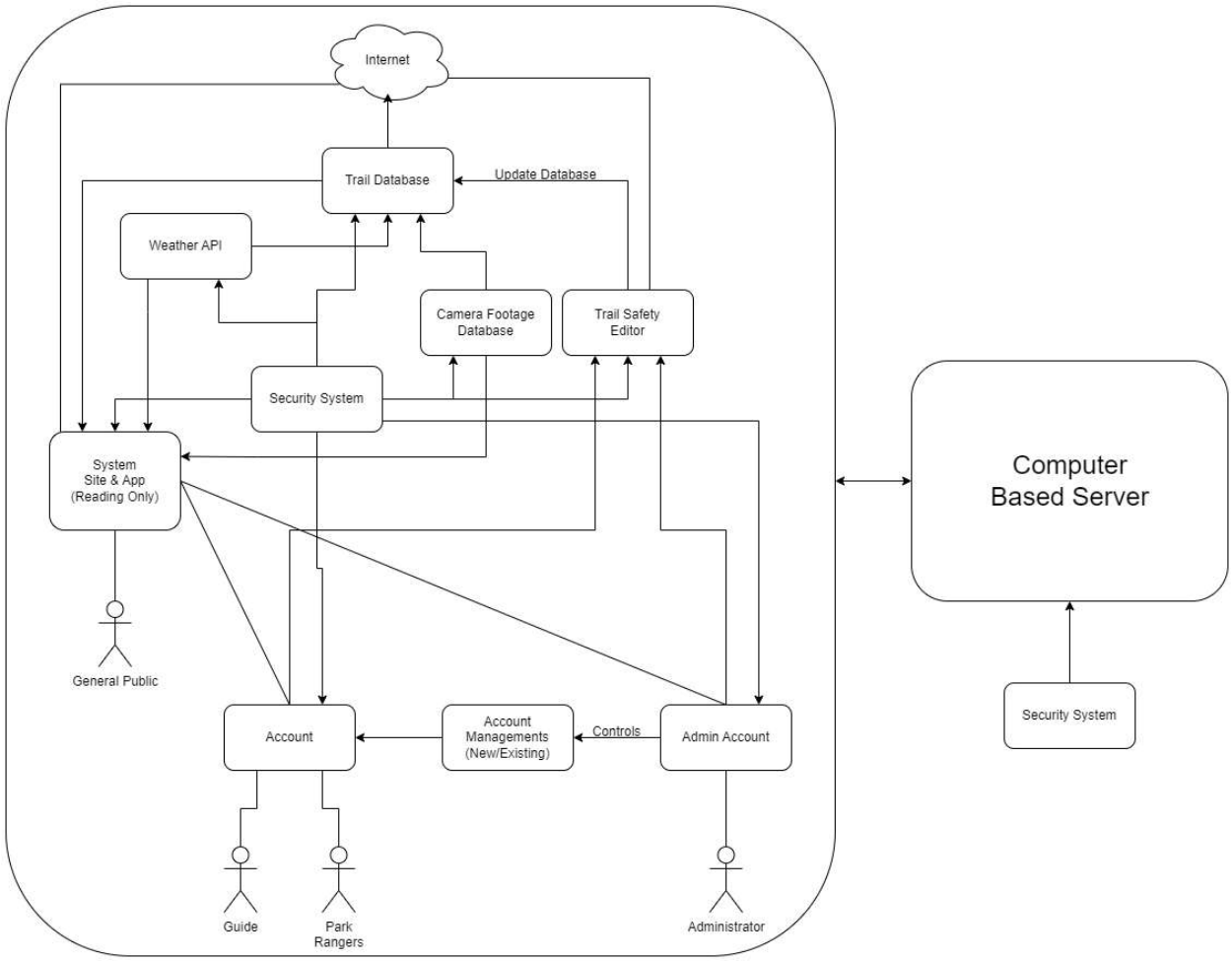
Prepared by Jackson McCoy, Samantha Konicek, Jien Zheng

Mar 4, 2023

System Description

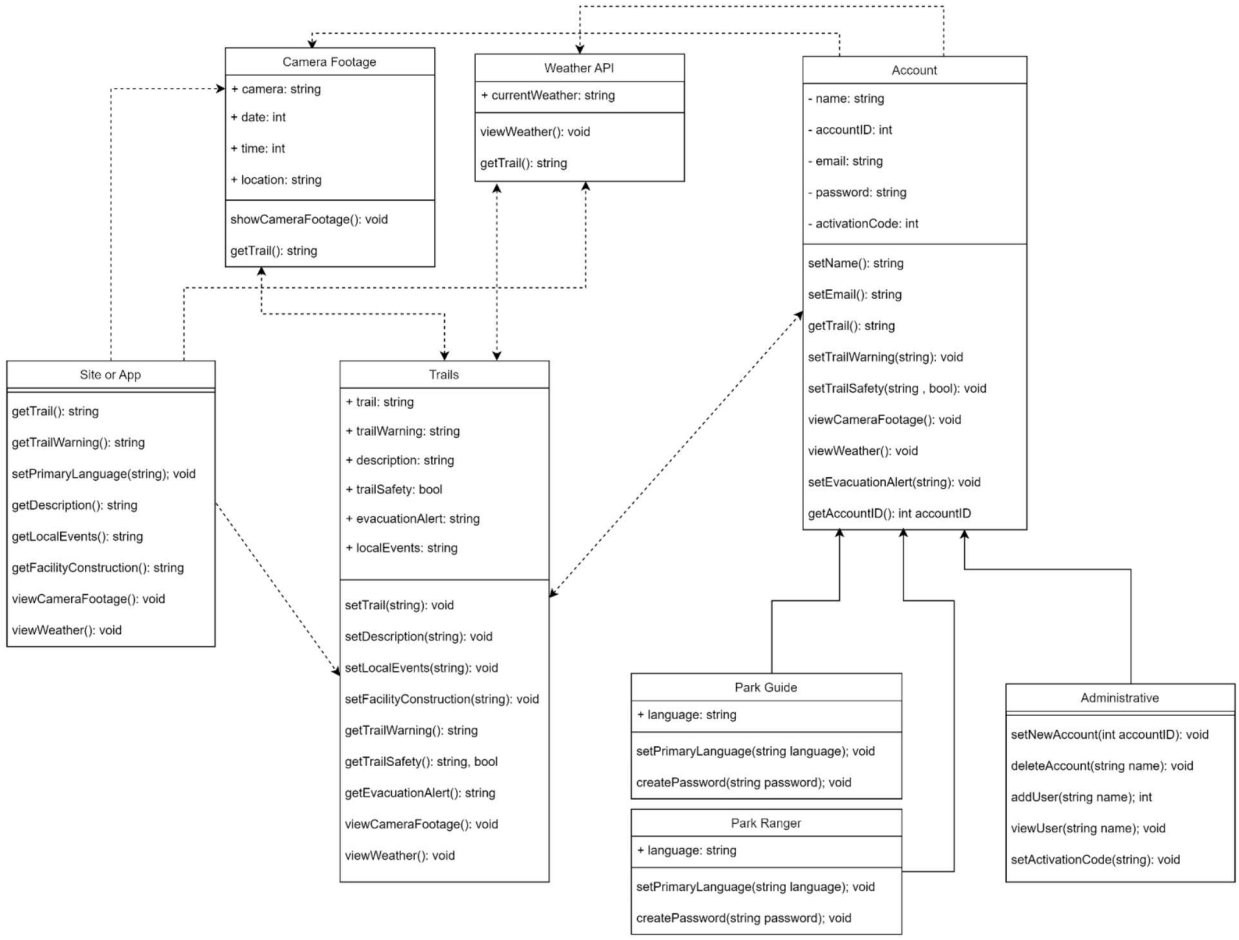
The Kilimanjaro Trail Monitoring System will be a robust and reliable system intended to provide important information on Kilimanjaro's many trails to park staff and guide leads in the area. The system will remove the guesswork as to which trails are currently safe and unblocked, allowing a smoother and less-dangerous experience for all. The system should also prevent freak accidents related to unsafe conditions by supporting evacuation alerts in the event of an emergency. This will be done by allowing guides, rangers, and park staff to mark the status of the trails of Kilimanjaro, which will be available to the general public when using the system. It is also important to note that Mount Kilimanjaro is located in the African nation of Tanzania. Therefore, the system will function on Tanzanian specifications.

Software Architecture Overview



This system is all under a computer-based server, where the computer at the main entrance of the park can take care of all the necessary tasks. There is a security system in place to receive protection against potential rebels/outsiders from changing information within the system. Inside the system itself, users with an admin account can edit the conditions of each trail within the system. They can check the conditions of each trail using the website or the application, giving them access to the weather conditions and camera footage at each trail. Admin users can additionally manage new accounts for park guides and park rangers by giving them an access code for the creation of an account. Park rangers and guides can, much like administrators, change the conditions of each trail within the system. They can check the conditions of each trail using the website or the application, which gives them access to the conditions and footage. Once the conditions of any trails are changed, a log of the changes is stored in a database of all the changes made and the current conditions of each trail, which ensures accountability and protects against malicious use. This also gives access to park rangers and guides to use the internet to change the conditions of the trails if needed. Another use of the database is in regards to camera footage, which is sent to the corresponding trail that

contains the correct camera. This footage allows users access to the cameras if they need to view trails. Weather API is used to determine the current weather conditions at each trail. It relays this information to the trail database and the website and application for users to access the current weather conditions. The general public can access the trail information using the internet to access the website or the application on their mobile device. The trail database can be accessed from the internet from any device that either uses a web browser or application.



The Park Guide and Park Ranger class both contain only one attribute: language. This attribute can be seen by anyone to show what is their primary language. They both contain the operations to set their primary language as well as the ability to create a password for their account. The Administrative class contains no attributes. It contains the operations of setting a new account, deleting an existing account, adding users to a list, viewing individual users, and setting an activation code to give to new users who are creating a new account. All three of these classes are considered Accounts, which is the parent class. Accounts have multiple attributes that are not able to be seen by others. These attributes are name, accountID, email, and password. AccountID should not be editable because everyone who is part of the organization will receive a unique ID for them to use to log into the system. Operations within the Account class contain getting the name of the trail, setting trail warnings, setting trail safety,

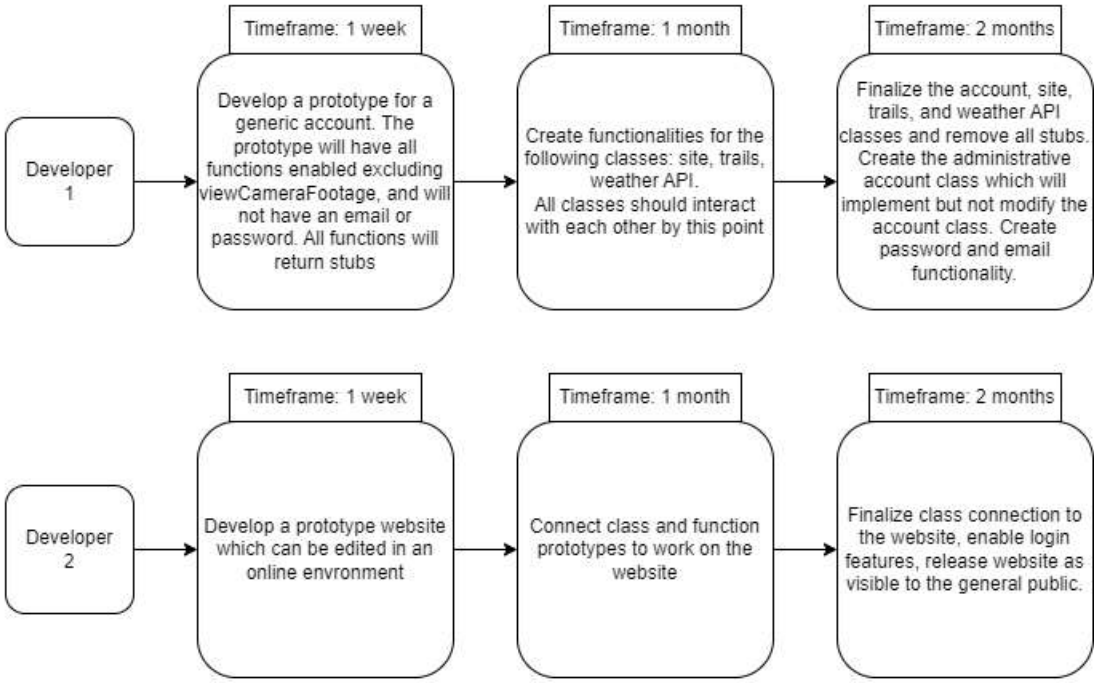
viewing camera footage of each trail, viewing the weather of each trail, setting evacuation alerts, and getting the accountID. Setting trail warnings, setting trail safety, and setting evacuation alerts are safety features that notify people around the Kilimanjaro area of the dangers present. The account class uses the Camera footage class. This class contains the camera's name, date, time, and location. This can be seen by everyone because it provides the public information about what is around each of the trails. Class Camera Footage has an operation that uses the Trail class to get the name of the trail that the camera is placed in. It has another operation that shows the camera footage at that certain trail if it is requested. The Account class uses the Weather API class to display the current weather conditions. This class shows the weather conditions if it is requested. It uses the Trail class to get the name of the trail so that it can provide the weather condition at that current trail.

The Trail class contains the trail attribute showing the different names of the trails. There are many operations in the Trail class. The trail class has setters to add the name of the trail, a description of the trail, local events, and facility construction. It also contains operations to view a trail's current warnings, if they have any, a trail's current safety condition, as well as receive any evacuation alerts that may be in effect. This provides users with information about what is around the trails and their descriptions. Additionally, there are also operations to view the current camera footage along with the weather of any given trail, done so by using the Camera Footage class and the Weather API class.

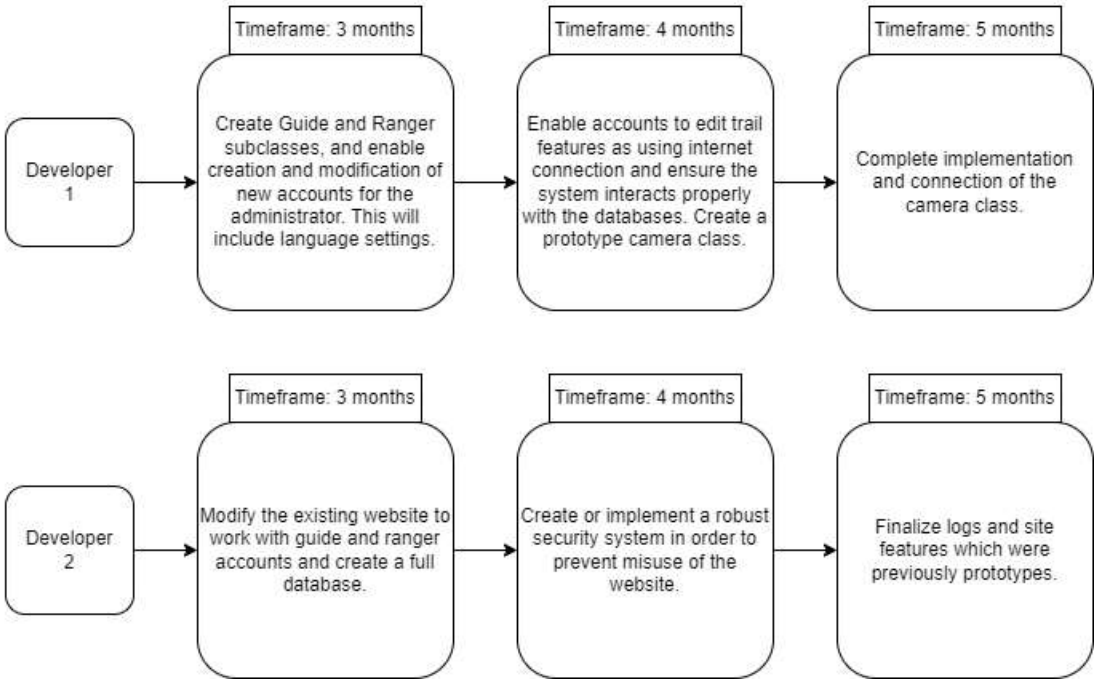
The Site or App class uses the Trail class, Camera Footage class, and Weather API class. It contains zero attributes as it is mainly for the public to read, and the public doesn't need to create an account or modify any trail attributes. It provides users with access to information about the different Kilimanjaro trails. This information includes the different trails available coupled with a description of the trail, any warnings currently being broadcasted about the trail, any events or celebrations scheduled to take place within the park, and any news on construction of new facilities within the park. There are also options available to view the current camera footage from the park as well as see current weather conditions.

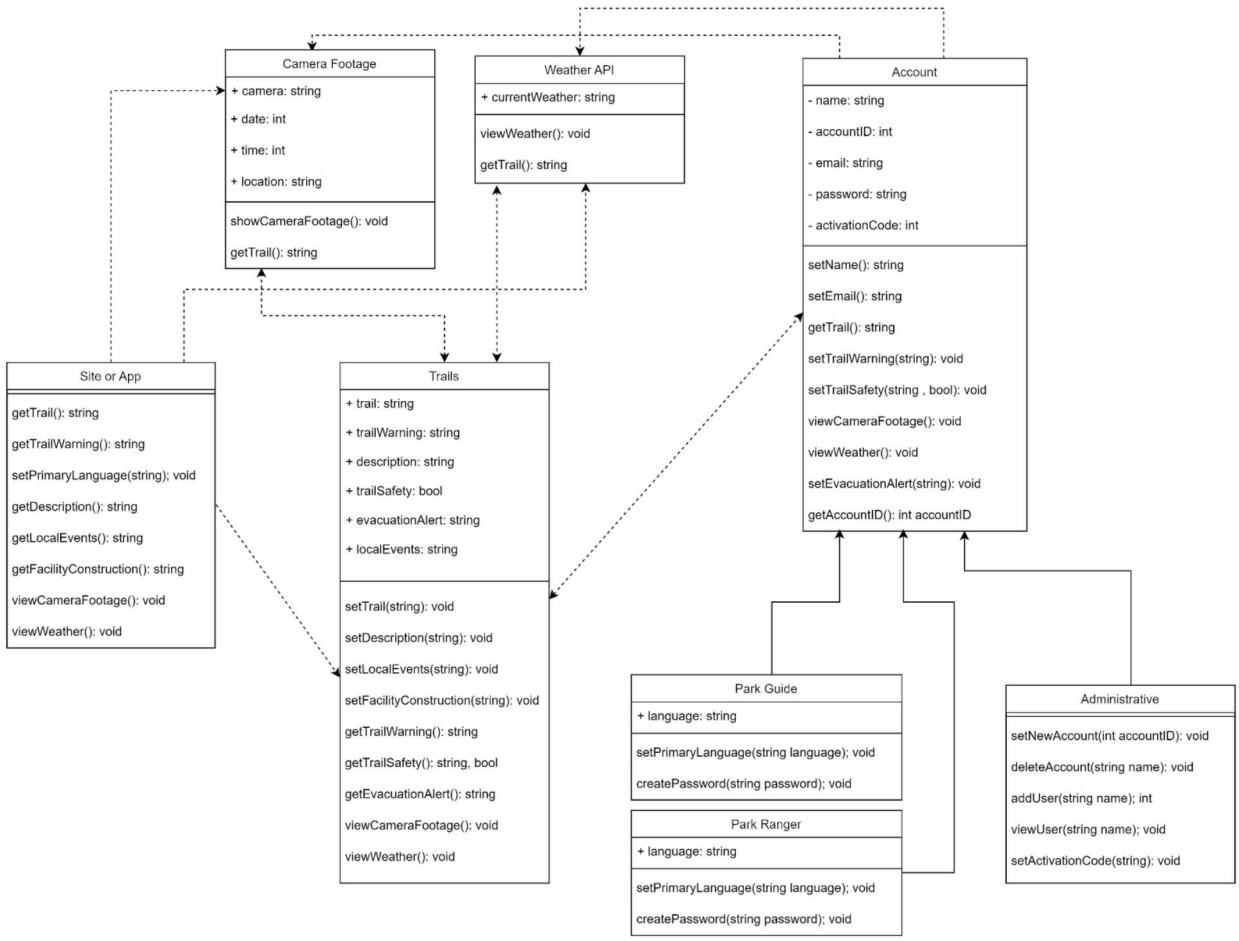
Development Plan and Timeline

Increment 1: Basic Interactive System For a Single User



Increment 2: Full System Compatibility and Cloud-Based Editing





Unit Tests:

Test 1: setPrimaryLanguage() function

accountname.setPrimaryLanguage(string language)

Testing the setPrimaryLanguage() function will ensure that new users are able to choose which language the site/app will appear in. Because the setPrimaryLanguage() function is copied into three different classes- provided that one instance of this function works, all should, making it an ideal test candidate. Due to the limit of languages that the system can display the only allowed languages, and therefore strings are “English”, “Spanish”, “Tagalog”, “Swahili”, “Chaga”, “Ngasa”, “Pare”, and “Gweno”. Other allowed strings would be the names of these languages in their own language, such as Español for Spanish
setPrimaryLanguage(string language)

1. Valid string (one of the languages that is allowed)
   Ex: user.setPrimaryLanguage(English)
   - Changes the language string stored in user to English
     Setting the primary language to English will set the user’s account to use English for all text.

2. Invalid string (any other input)
   Ex: user.setPrimaryLanguage(OogaBooga)
   - Tells the user that OogaBooga is not a supported language
     Because OogaBooga is not a supported language, the system should not mark the language as OogaBooga since it will not work. Hence, trying to set the language to OogaBooga will inform the user that they cannot do this, and prompt them to input a different language.

Test 2: setDescription() function

trailname.setDescription(string description)

Providing basic facts about each one of Kilimanjaro's trails is a priority for the system, thus requiring the setDescription function. However, it is important to note that the setDescription function should be a shorter description and not a full article about the trail. Therefore the length of the description should be under or at 100 words in order to improve readability. Any string under or at 100 words should validate, while any string over that number of words should not be allowed by the system.

1. String under the character limit
   Ex: trailA.setDescription("A mostly-flat trail perfect for family outings. 5.5 mile loop.")
   - The description attached to trailA is now the above input. This means that when a user activates the getDescription function they will now see this text.
2. String above character limit
   Ex: trailA.setDescription(testCase) [where test case is a string containing the following sequence 25 times: "This string is too long "]
   - The description for trailA is not changed and the system informs the user that their description was too long.

Integration Tests:

Test 1: setTrailWarning and getTrailWarning integration

trail.setTrailWarning(string trailWarning) void

trail.getTrailWarning(void) trailWarning

This function allows the user to set a warning to go along with a given trail if there are any current dangers or potential dangers involving the Kilimanjaro trails. This feature is very important and must be working properly, as it's how users and people venturing out on the trails will be provided with any information on things they should be cautious of while in the park. This function is located in the accounts class, as park rangers with accounts are the ones who have access to set and update these warnings. Since the trail warning can be accessed from two other classes, the trail class and the site/app, it's a good way to test the integration of the system.

1. Extreme Weather
   Sample Input: trail.setTrailWarning("Warning: Extreme weather conditions. Caution is advised.")

Expected Output(for getTrailWarning()): Warning: Extreme weather conditions. Caution is advised.

- The user(ranger) sets a warning for extreme weather currently in effect at the trails which is then accessible by external users, with the expected output being the string which was input.

2. No Current Warning

   Sample Input: trail.setTrailWarning("No current trail warnings")

   Expected Output(for getTrailWarning()): "No current trail warnings"

   - This would be the default setting when there are currently no warnings active for the trails.

3. Null Input

   Sample Input: trail.setTrailWarning(null)

   Expected Output: "Error: null"

   - If no string is entered, a null error will appear and a new trail warning will need to be set by an account user.

Test 2: Camera Usage

account.viewCameraFootage(string camera): void

The camera functionality is very important for the trail system. The users need to be able to access the camera placed on top of the mountain in order to view the current weather and trail conditions in the park. This is a key part of the system as this information is then used to implement any applicable trail warnings or evacuation notices, using account.setTrailWarning(string trailWarning) or account.setEvacuationAlert(string evacuationAlert), that park visitors and other park rangers / guides would need to be aware of. If the camera isn't properly interacting with the system, crucial information wouldn't be provided and could result in dangerous situations for park guests. The function is called from the account class and uses the camera attribute from the Camera Footage class.

1. Camera isn't connected / isn't working properly

   Sample Input: account.viewCameraFootage("KilimanjaroCamera")

   Expected Output: "Error: KilimanjaroCamera is not connected. Check the connection and try again."

   - The camera atop the mountain isn't properly connected and isn't providing current footage. It should be ensured that extreme weather conditions don't cause connection problems, as there can be extreme weather on the mountain and this shouldn't impact the functionality of the system.

2. Camera is connected and working properly

   Sample Input: account.viewCameraFootage("KilimanjaroCamera")

   Expected Output: "Current Footage:" coupled with live video from the mountain

   - In this scenario, the camera is working correctly and the camera footage is being received from the mountain, showing the current weather and conditions of the trails.

System Tests:

Test 1: Account Creation

Testing the new account creation requires an admin to be able to provide an activation code for each time the user logs in to the website. When a new park ranger or park staff has to create a new account, a screen will be prompted to ask for the name, username, email address and password. Once the account is created, it will send the accountID via email, so that the user will be able to log in with the accountID or the email address. Each time the user logs in to the system, an activation code will be needed alongside the username or email address and password. After logging into the system, the user will be able to access various commands. They are able to get the names of each trail, creating a warning for a specific trail, setting if a trail is either safe or not safe, viewing camera footage and weather, creating an evacuation alert, setting the primary language, and getting their accountID.

1) Details needed for account creation

Example 1) *account.setName(String)*

Input = "Casper Keith"

Expected Output = Name is a member of company

- The user must be a park ranger or a park staff in order for them to be able to create an account. They must use their real names that are written on their birth certificate to create an account. This is so that the company can keep count on who is a member.

Example 2) *account.createPassword(String)*

Input = "norMalLemon"

Expected Output = Password does not meet requirements

Input = "n0rMa!Lemon"

Expected Output = Password meets requirements

Input = "n0rMa!Lemon1"

Expected Output = Password does not match

Input = "n0rMa!Lemon"

Expected Output = Password matches

- When the user is creating a password for their account, it requires at least one capital letter, one number, and one symbol. Once the password meets all the requirements, the user needs to confirm their password again to make sure it is the password that they will be using.

Example 3) *account.setEmailAddress(String)*

Input = "casperk293@gmail.com"

Expected Output = Email accepted

Input = "casperk293@gmail"

Expected Output = Email not accepted

- Users will need to put in a personal or work email that they will be using for work. The user must have the domain, and it will check to see if the email is valid, so that the account can be created. An email will be sent to the email address confirming that an account

has been created. It will also send an accountID within the email, so that the user can use the userID to log in.

2) Checking if the account was successfully created
    Example 1) *account.viewUser(String)*
        Input = “Casper Smith”
        Expected Output = No user with the name “Casper Smith” is found
        Input = “Casper Keith”
        Expected Output = User with the name “Casper Keith” is found
        The given accountID will be 926394739 as an example.
        Input = “926394729”
        Expected Output = No user with the accountID “926394729” is found
        Input = “926394739”
        Expected Output = User with the accountID “926394729” is found
        Input = “casperk823@gmail.com”
        Expected Output = No user with the email “casperk823@gmail.com” is found
        Input = “casperk293@gmail.com”
        Expected Output = User with the email “casperk293@gmail.com” is found
        - An administrator is needed to check if the account was created. It could use the name, accountID, or the email address to confirm the user has an account in the system.
3) Logging into the system
    Example 1) *For this example;*
        An Activation Code will be 8236, Account ID will be 926394739, and password will be n0rMa!Lemon.
        Input = “AccountID - 926394739, password - n0rMa!Lemon, Activation Code - 8336”
        Expected Output = The Activation code did not match the system
        Input = “AccountID - 926394729, password - n0rMa!Lemon, Activation Code - 8236”
        Expected Output = The accountID is not found
        Input = “AccountID - 926394739, password - n0rMa!Lemon1, Activation Code - 8236”
        Expected Output = The password is incorrect, Try again
        Input = “AccountID - 926394739, password - n0rMa!Lemon, Activation Code - 8236”
        Expected Output = Log in successful
        - An activation will be generated from the Administrative account. The only way to get the activation code is by talking to the administrator and asking them for the activation code. The user also must enter the correct information as well as the activation code in order to get access to their account.
4) Testing accessible features
    Example 1) *account.getTrail()*

- Users are able to use the feature get trail. This allows the user to get a list of all the trails with their names. If the user were to push the button that states “Trails”, it would be sent to a page with a list of all 12 trails that are listed in the system.

Example 2) *account.setTrailWarning(String trailWarning)*

- Users are able to create a warning that can be sent out to all park staff and rangers, and be displayed on the front page. They will have to click on the button that says “Create Warning” and it will take them to a page where they are able to write the warning to be sent out.

Example 3) *account.viewCameraFootage()*

- Users are able to view the different camera footage of each trail that they want to look at. They will have to click on the button that says “Camera Footage” and it will take them to a list of trails with the different cameras and the past camera footage of each trail.

Example 4) *account.viewWeather()*

- Users are able to view the current weather of each trail. They will have to click on the button that says “Weather” and it will take them to a page listing the different weather conditions that will be present at each specific trail.

Example 5) *account.setEvacutationAlert(String evacuationAlert)*

- Users are able to create an evacuation alert that will be sent out to all park staff and rangers, and be displayed on the front page. They will have to click on the button that says “Create Evacuation Alert” and it will take them to a page where they can write the evacuation alert to be sent out.

Example 6) *account.setPrimaryLanguage(String language)*

- Users are able to set their primary language for their user side experience. This allows them to set which language that the site/app will appear for them because it is for their personal use and preference.

Example 7) *account.getAccountID(Int number)*

- Users are able to get access to their accountID if they have forgotten their ID because they could have used their email address to sign in. They will have to click on the button that says “Account ID” and it will bring in a pop up message with their accountID.

Test 2: Mark Trail as safe

Marking a trail as safe from a systemwide point of view requires a user account. The user will need to log into their account, select the trail they wish to mark as safe, confirming this. The system should then display the selected trail as safe, logging the action in the database. It is also of note that a trail cannot be set as safe if it is already safe, but since this condition should have no output or changes to any variables whatsoever it is not notable for testing.

1) Logging into the system
   For this example the Account ID will be 926394739, and password will be n0rMa!Lemon.
   a) *Input = “AccountID - 926394729, password - n0rMa!Lemon”*
      Expected Output = The accountID is not found
   b) *Input = “AccountID - 926394739, password - n0rMa!Lemon1”*

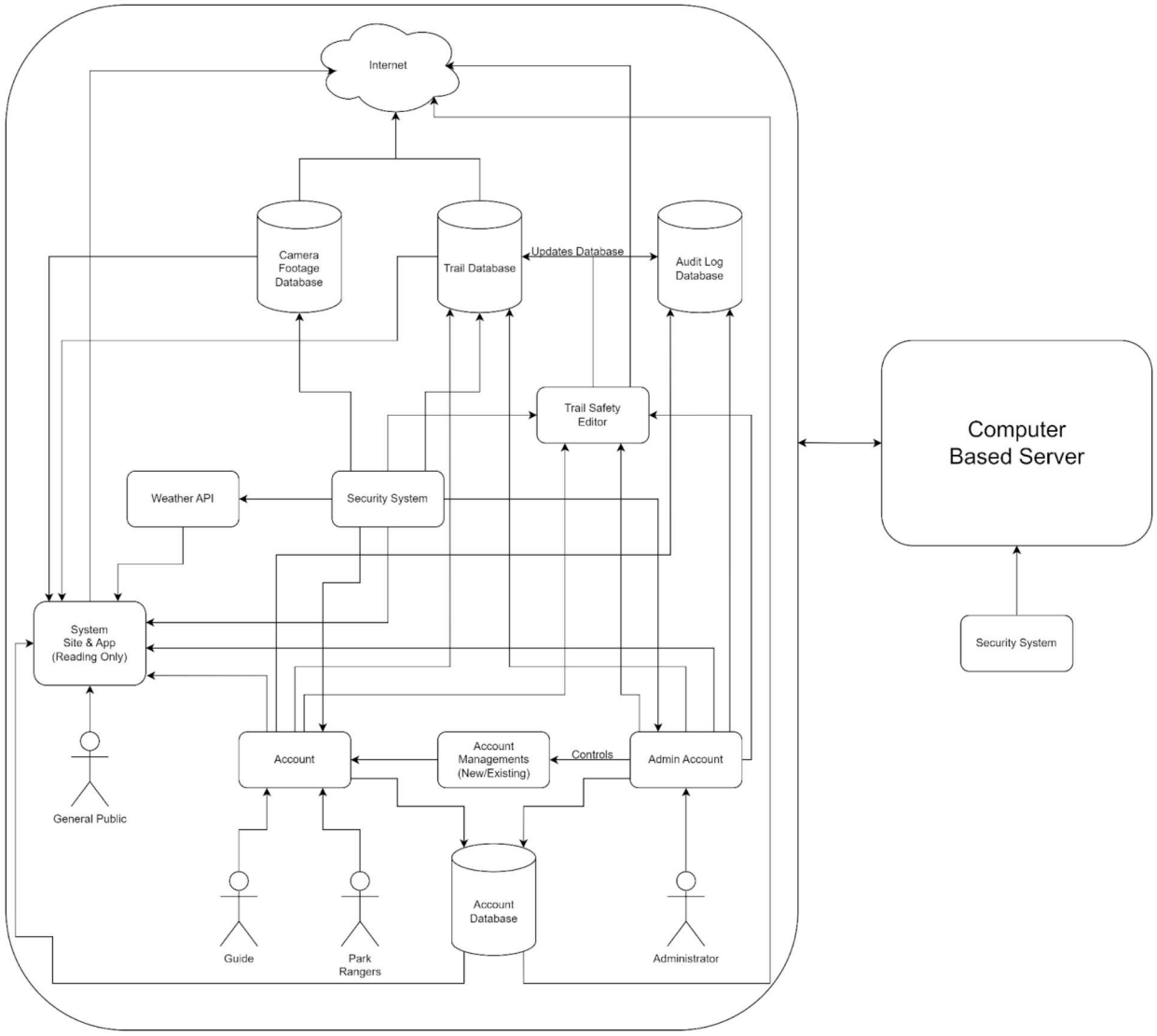
Expected Output = The password is incorrect, Try again

c) *Input = "AccountID - 926394739, password - n0rMa!Lemon"*
Expected Output = Log in successful

2) Selecting the desired trail and Marking the safety of the trail as safe
The desired trail would be selected by the user and when marked as safe would call the function setTrailSafety(string trailName, bool safetyStatus). For testing purposes, we assume the name of valid trails is trail1 through trail12, all named similarly. The user needs to make sure they are entering the proper boolean, as using the boolean of 1 and not 0 will set the trail to be unsafe instead. This output, while functioning, is not what is wanted
   a) *Input = "trail1, 1"*
   Expected Output = Please confirm that trail1 is now unsafe.
   b) *Input = "trail13, 0"*
   Expected Output = Trail 13 does not exist. System cannot proceed.
   c) *Input = "trail11, 0"*
   Expected Output = Please confirm that trail11 is now safe.
3) Confirming trail safety
The user will appear to run the same function, setTrailSafety(string trailName, bool safetyStatus). However, they are still inside the same function, so it is assumed that they have input "trail11, 0", as that was the successful test from the previous step
   a) *Input = "trail11, 1"*
   Expected Output = Trail status does not match. Trail status was not confirmed.
   b) *Input = "trail13, 0"*
   Expected Output = Trail does not match. System cannot proceed..
   c) *Input = "trail11, 0"*
   Expected Output = Trail 11 has been confirmed safe.
4) Displaying safety for the trail set
Users will now have to return to the homepage of the system to confirm the safety of the trail and make sure that the trail is displayed correctly. The user must click on the trail that they have marked as safe or unsafe. This would call the function getTrailSafety(string trailName, bool safetyStatus). If there is a trail that has not been marked as either safe or unsafe, it will produce the outcome as "trail # safety is unmarked." Otherwise it will display the trail as either safe or unsafe.
For this test, we assume that trail 11 is marked as safe, and trail 1 is not marked at all.
   a) Input = "trail11"
   Expected Output = Trail 11 is safe.
   b) Input = "trail1"
   Expected Output = Trail 1 safety is unmarked.
   c) Input = "trail13"
   Expected Output = Trail 13 is not found.
5) Logging changes to the database
Following the completion of the trail safety, the username and account ID of the account that made the changes are logged to the external database. Though this is not directly integrated into the system, in the case of the inputs above, the account with the

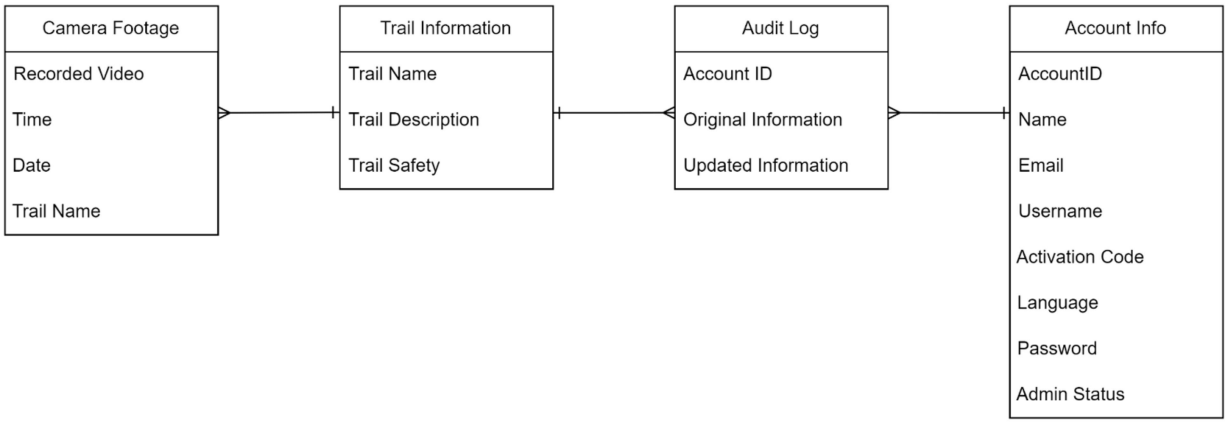
AccountID of 926394729 *and* email of casperk293@gmail.com *will be sent to the database.*

Software Design 2.0



From the original software architecture diagram, four different databases were added in support of changes to how the data is stored. Some minor changes include the lines are more spread out as well as the whole diagram itself. This increases the readability of the diagram as well as more clearly representing the flow of data into the databases. This has also caused some changes to the trail database, which will now contain actual trail information instead of the audit log information it previously held. The audit log will become a separate database which will essentially take the place of the old trail database. In addition to the audit log, the new databases will store information from the camera footage on each trail and which account made changes. The account database contains information about each account and their public information like their name and email address. Allowing the public to see the name and the email address of the employees can help the employees from people informing the employees

of possible dangers within the trails. Each employee can see certain information from the account database with some being private for their security.



For this system, SQL database design has been chosen as the most effective method of storing the information. There were several reasons behind this choice. First of all, the SQL model provides a consistent structure for databases that will not be going through large scale transitions. As the information stored is very unlikely to need to have expanded fields or need an increase in the number of databases, there is no need to complicate the process. Since the data is not hierarchical, the SQL databases will not struggle to store it. Lastly, the ACID properties, while not necessarily critical for this system, are an added bonus of reliability.

There are four databases that will be used in this system: camera footage, trail information, audit log, and account information. Each of these were chosen as separate databases to maximize information while minimizing overlap. Originally, it was clear that there would need to be at least two databases; one for trails and one for accounts. This is because the trails and accounts are only associated when making audit logs, so they have almost no common use. The audit log was split from the trail category in order to keep the trail information uncluttered and ensure that the audit log is kept to only storing changes to trails, not the trails themselves. Though this created some overlap in storing the most recently updated information which could also be accessed through trail information, it was judged to be acceptable to allow proper storage of the changes made to trails. Finally, camera footage was separated from trail information due to each trail storing multiple instances of footage. If this was stored in the trail information database, it would require a corresponding increase in the amount of data stored but without the flexibility of a fourth database. We also considered adding a fifth database for admin accounts, but this was judged to be too much overlap with account info, and was instead added as an additional field.

The relationships between the different databases became quite clear following their splitting. As stated above, it was known that the trail information and account info databases would connect with the audit log in some fashion, while the camera footage would need the trail information to fully function. What was less clear was the nature of these relationships, and several different types were proposed before the final model was completed. Trail information would have a one-to-many relationship because each trail has multiple video recordings, but each recording has just a single trail. The camera footage would use trail information. Trail information also has this relationship with audit log, for a similar reason, and the audit log would

use trail information. Account info also has a one-to many relationship with trail information, which is a bit less clear. While the audit log itself would have many accounts, each specific log can only have one account creating it. On the other hand, each account could have multiple log entries, thus being a one-to-many relationship. This would be described as an account creating a log entry.

Within each of these four databases, different sets of fields are stored. In the camera footage database there are the following fields: recorded video, time, date, and trail name. Recorded video, which is the recorded footage from the different trails is saved as files. Alongside these files, the time and date of the recorded footage are stored as doubles, and the trail name corresponding to the footage is stored as strings. The trail name field is also included in the trail information database. Alongside this, there exists the field of trail description, which contains a brief description of what the trail is like, and as well as safety, which describes the current safety conditions of the trail, both of which are stored as strings. Trail safety would be saved as a boolean, for safe and unsafe, but due to having a special case for when the whole park is closed, has to be stored in a different manner. The third database is the audit log, which keeps track of changes that are made within the system. This database has three fields. It stores the new information that a ranger entered in the updated information field and the information that was in place before the update in the original information field. Both of these fields are stored as strings. Additionally, this database also contains the account ID field so that users are able to tell who made the updates in case there are any problems. Account ID is stored in the fourth database, account info, and is stored as an integer. The language, email, password, name, and username fields are all stored in this database as strings. Additionally, the activation code, used when first creating an account, is stored as an integer. Admin status for the account is stored as a boolean; true if the account is an admin and false if otherwise.