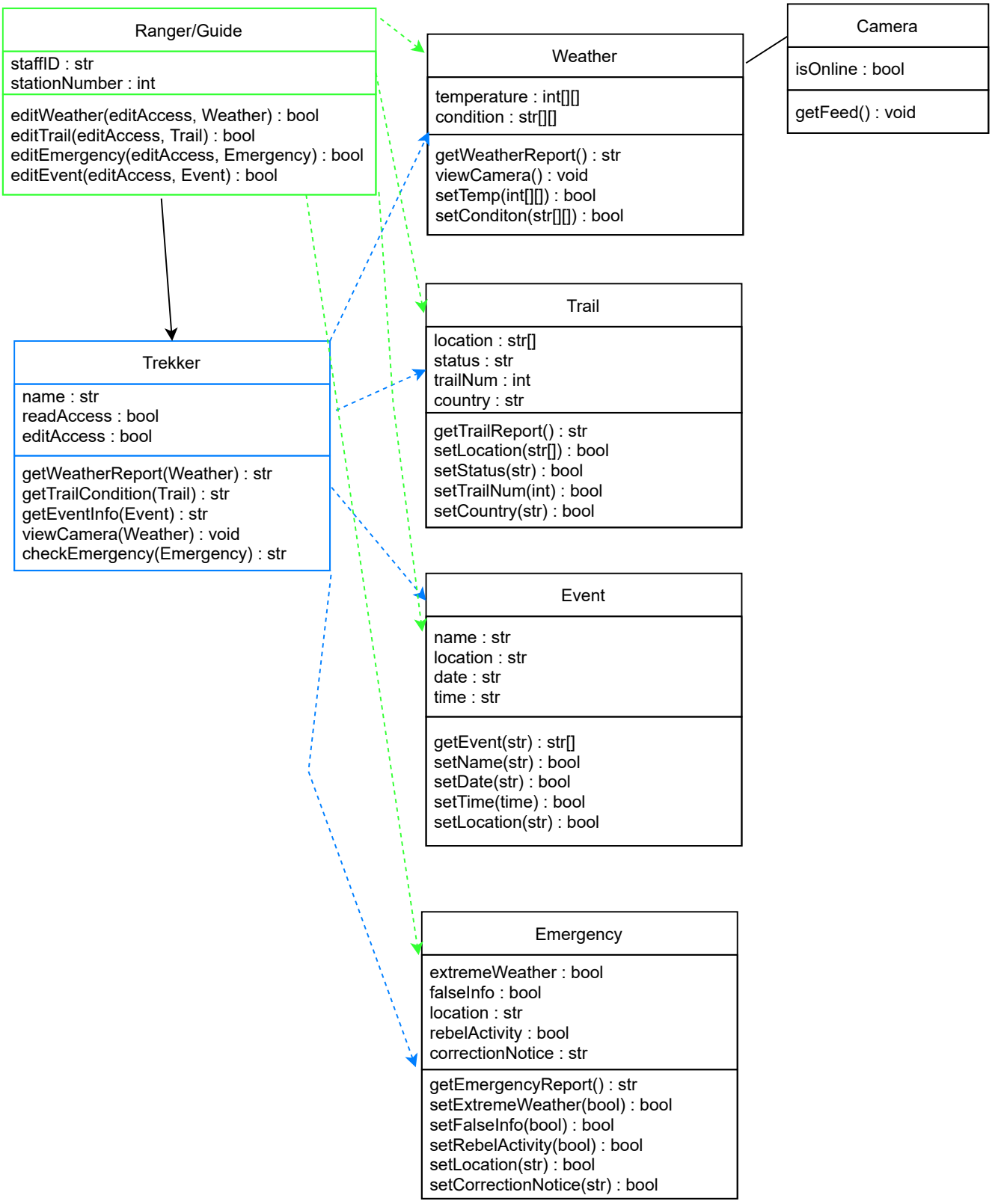
Kili Trekker System

Group Members : Dante Viscuso, Eduardo Barraza, MichaelJames Gurtiza, Animesh Srivastava

Kili Trekker System Description

This software system is going manage to all relevant and real-time information about the 12 available trails located at the Kilimanjaro National Park. Relevant trail information includes weather reports, trail conditions, ongoing park events and any rebellious or malicious activity within the park. This system will work in tangent with the already in place cellphone towers scattered across the park in order to connect all of the ranger stations to the main server. User of this system should be able to interact with and view all information as needed inside and outside the national park. It will be the park rangers' and guides' job to keep the systems information up to date via ranger station or web-based cell phones. This system is being implemented with trekkers, guides, and park rangers in mind in order to provide a smooth experience through the trails, have information readily available, and provide a safe experience through various alerts of rebel activity and correction of incorrect information posted on the system. This document is meant to outline the various requirements that must be considered in order to create this system. The requirements are laid out in various sections: such as the user requirements, system requirements, and other relevant information and these sections will contain information about how this system works, the features it will include, examples of how different users might interact with this system, as well as an overview on how this system was created and further updated.

Kili Trekker System UML Class Diagram



Class: Trekker

This is the basic user class which will determine if they can read/edit and keeps track of the user's name in order to keep track of all trekkers in the park. The class will outline all the functions that can be performed by the average user.

Attributes:
readAccess : bool – true or false variable that will determine if they can read the information from the system (always true)
editAccess : bool – true or false variable that will determine if they can edit the information of the system (false for the basic user class)
name : str – holds the name of the user to ensure they are a verified park goer

Operations:
getWeatherReport(Weather) : str – uses the weather class in order to generate a weather report in the form of a string
getEventInfo(Event) : str – uses the event class in order to generate an event report in the form of a string
getTrailCondition(Trail) : str – uses the trail class in order to check the various trails and any current conditions with them in the form of a string
checkEmergency(Emergency) : str – uses the emergency class to check if there are any ongoing events, returns this info in the form of a string
viewCamera(Weather) : void – uses the weather class in order to connect to the mountain camera for a live feed view, returns nothing

Relationships:
Trekker uses Weather for the weather report and camera view
Trekker uses Event for information on current events
Trekker uses Trail for the trail conditions
Trekker uses Emergency to star ware of any ongoing emergencies

Class: Ranger/Guide

This is the ranger/guide class which extends the basic 'Trekker' class in order to give it specific functions unique to only this class. The class will outline all the functions that can be performed by the rangers and guides of the trails.

Attributes:
staffID : str – an ID or badge number to keep track of which ranger or guide is in use of this class
stationNumber : int – this indicates which ranger station they are currently located at

Operations:
editWeather(editAccess,Weather) : bool – setter-like functions which allows the ranger/guide to input or edit the data relating to the weather class, such as temperature or the overall condition, returns true when successful
editEvent(editAccess,Event) : bool – setter-like functions which allows the ranger/guide to input or edit the data relating to the event class, such as a future event along with its name, returns true when successful

editTrail(editAccess,Trail) : bool – setter-like functions which allows the ranger/guide to input or edit the data relating to the trail class, such as the trail number along with its current status, returns true when successful
editEmergency(editAccess,Emergency) : bool – setter-like function which allows the ranger/guide to input or edit the data relating to the emergency class, such as if there is an emergency regarding harsh weather or rebel activity, returns true when successful

Relationships:
Ranger/Guide uses Trail to edit the trail conditions
Ranger/Guide uses Weather to input the current weather
Ranger/Guide uses Events to edit and input any information about events at the park
Ranger/Guide uses Emergency to update the system of any ongoing emergencies to keep the trekkers safe
Ranger/Guide is a Trekker class so it can do everything a trekker can, with the added edit functions

Class: Weather

This is the weather class which will contain all current and future information about the weather conditions. This class will outline the various weather-related function and the ways a user might interact with this class.

Attributes:
temperature : int[][] – a 2D array that holds the temperature of each day, and withing the days, an hour-by-hour forecast
condition : str[][] – a 2D array that holds the condition of each day, and withing the days, hour-by-hour status

Operations:
getWeatherReport() : str – basic function that generates a weather report in the form of a string by using its private class variables
setTemp(int[][]) : bool – setter function that will allow the input or editing of temperature at Mount Kilimanjaro National Park, returns true if change is successful
setCondition(str[][]) : bool – setter function that will allow the input or editing of weather conditions, returns true if change is successful
viewCamera(Camera) : void – connects to the live feed camera by using the camera class

Relationships:
Weather is associated with the camera class to be able connect to the mountain camera for a live feed view
Weather is used by Ranger/Guide class to edit the class's information through its functions
Weather is used by the Trekker class to view a weather report

Class: Trail

This is the trail class which will contain all current information about the trail conditions. This class will outline the various trail-related function and the ways a user might interact with this class.

Attributes:
location : str[] – a string array that will hold the geolocation of the trail from start to finish
status : str – a string that hold the current status of the trail

trailNum : int – an integer that will distinguish which of the 12 trails is being referenced
country : str – a string that will hold which country the trail belongs to

Operations:
getTrailReport() : str – basic function that generates a trail report in the form of a string by using its private class variables
setLocation(str[]) : bool – setter function that will allow the input of coordinates of trail as a string array, returns true if change is successful
setStatus(str) : bool – setter function that will allow the input of the status of the trail as a string, returns true if change is successful
setTrailNum(int) : bool – setter function that will allow the input of a unique identification number for a trail as integer, returns true if change is successful
setCountry(str) : bool – setter function that will allow the input of which country the trail belongs to as a string, returns true if change is successful

Relationships:
Trail is used by Ranger/Guide subclass to edit the class's information through its setter function
Trail is used by the Trekker class to view a trail report of a given trail

Class: Camera

This is the camera class in which its sole purpose is to be called through the weather class by the trekker in order to show a live feed view of the trails. The class will outline the one function that is performed in this class.

Attributes:
isOnline : bool – true or false variable that will determine if the camera is online and ready

Operations:
getFeed() : void – simple function which is called by the weather class in order to connect to the mountain top camera

Relationships:
Camera is associated with the Weather class because the Weather class 'has a camera' in order to show a live weather feed
Camera is used through the weather class by the user to again show a live weather feed

Class: Emergency

This is the emergency class that will be responsible for dealing with any and all emergencies that occur within the park. This class will outline the different functions that can be performed by the trekker and the ranger/guide.

Attributes:
extremeWeather : bool – true or false variable that will determine if there is extreme weather within the area
falseInfo : bool – true or false variable that will determine if there is false information that has been inputted in the system
rebelActivity : bool – true or false variable that will determine if there is ongoing rebel activity within the area

location : str – a string variable that will hold the location of the emergency
correctionNotice : str – a string variable that will hold what the false information is and where is currently being shown

Operations:
getEmergencyReport() : str – uses the emergency class so that the trekker can see the emergency report in a form of a string
setExtremeWeather(bool) : bool – uses the emergency class so that the ranger/guide can set any emergency report, returns true when successful
setFalseInfo(bool) : bool – uses the emergency class so that ranger/guide can report if any info are false, returns true when successful
setRebelActivity(bool) : bool – uses the emergency class so that the ranger/guide can report any rebel activities, returns true when successful
setLocation(str) : str – uses the emergency class in order to set the location of the emergency in a form of a string
setCorrectionNotice(str) : bool – setter function to allow the input of a string to change correctionNotice, returns true when successful

Relationships:
Emergency is used by Ranger/Guide subclass to edit the class's information about any ongoing emergency through its setter function
Emergency is used by the Trekker class to check if there are any emergencies in the park

Class: Event

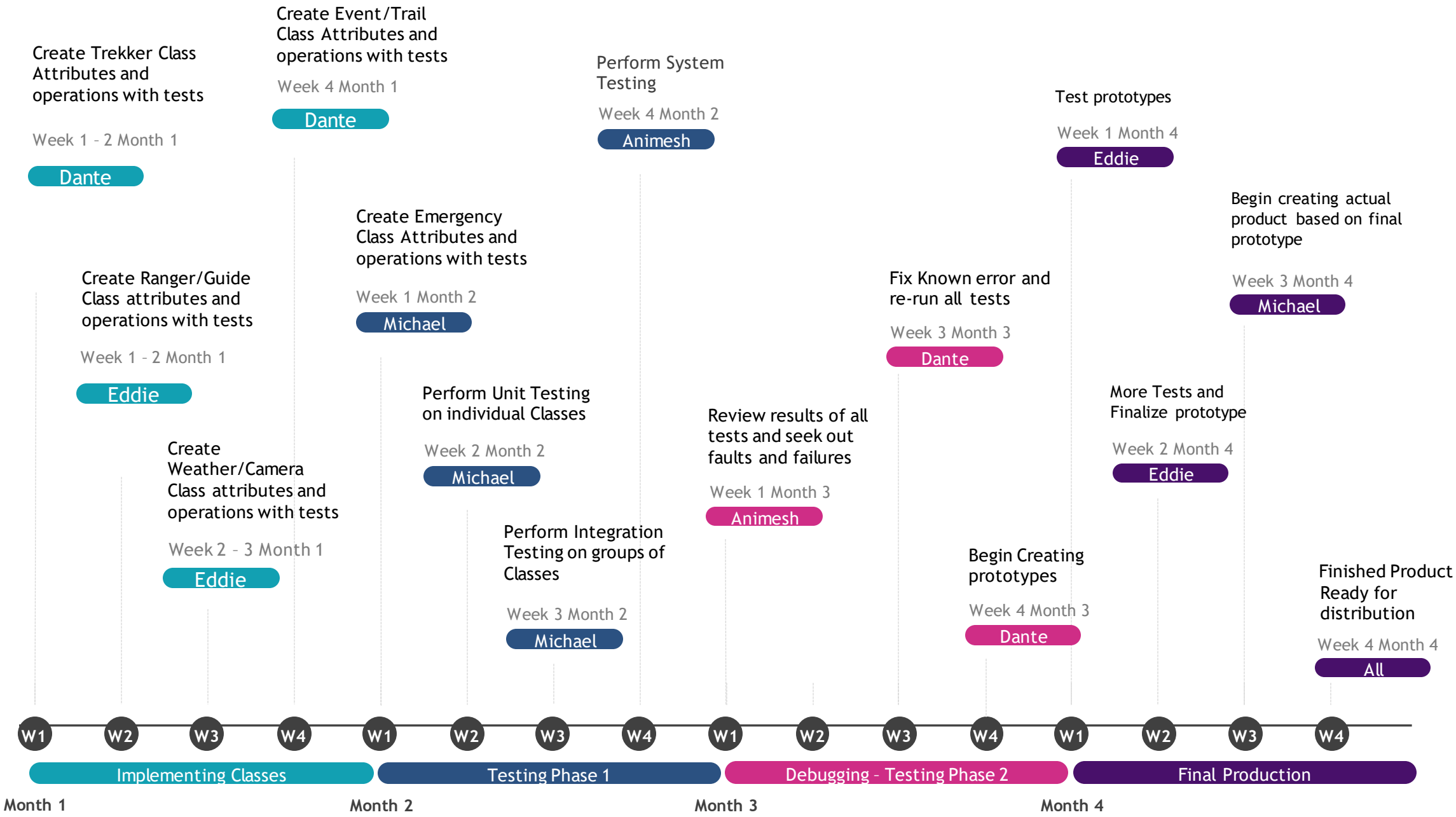
This Event class will be able to let trekkers know of information about events at the park. This class will also be able to let the Ranger/Guide edit/set any event that is happening at the park as well.

Attributes:
name : str – a string variable that will hold the name of the event
location : str – a string variable that will hold the location of the event
date : str – a string variable that will hold calendar date in which event occurs
time : str – a string variable that will hold the time of day event occurs

Operations:
getEvent(str) : str – uses the event class with string parameter to generate the event name in the form of a string
setName(str) : bool – uses the event class in order to set the name of the event in the form of a string, returns true when successful
setDate(str) : bool – uses the event class in order to set the date of the event in the form of a string, returns true when successful
setTime(str) : bool – uses the event class in order to set the time of the event in the form of a string, returns true when successful
setLocation(str) : bool – uses the event class in order to set the location of the event in the form of a string, returns true when successful

Relationships:
Trekker uses Event to determine the name, time, and location of the event
Ranger/Guide uses Event to set the name, date, time, and location of the current event

Kili Trekker System Timeline Nov 2021 - Feb 2022



SDS: Test Plan – Kili Trekker System

Unit Tests

Testing one component of the system for each test

In our test plan, we want our unit tests to mainly cover the class functions of the information holding classes such as the Event class, the Trail class, or the Weather class in order to ensure that the data is not only up to date, but also valid.

1. **setTrailNum(int) : bool**
   Invalid input: any number less than 1, any number larger than the total amount of trails (in this case it would be 12, null input
   - × {-1, false}
   - × {0, false}
   - × {“”, false}
   - × {234, false}

   Valid input: any number 1-max number of trails (at this time it is 12, but could increase)
   - ✓ {1, true}
   - ✓ {4, true}
   - ✓ {(max number of trails), true}

2. **setLocation(str) : bool**
   Invalid input: any data type other than an array of strings. Strings that do not contain a comma separating numbers will be invalid. First number shall not be greater than 90 nor lesser than -90, representing latitude. The second number shall not be greater than 180 nor lesser than -180, representing longitude. The precision of these numbers shall not be anything other than 5 decimal places; allows for the use of the decimal degrees system used to represent GPS coordinates with about a precision of 1.11 meters. Adjacent coordinates shall not differ by more than 0.00010, 0.00010. Letters inside the string are invalid. Array shall not have less than two strings.

   - × {{“-99.36283, 181.38263”,…, “-99.36193, 181.38353”}, false}
   - × {“-40.37263, 20.37253”, false}
   - × {{“43.38291m, 34.28392m”,…, “43.38256m, 34.28451m”}, false}
   - × {{1.38291, 4.28392,…, 1.38351, 4.28304}, false}

   Valid input: an array made of strings. Strings contain 2 numbers separated by a comma. First number can be anything from -90 to 90, while second number is -180 to 180. Numbers shall have a precision of 5 decimal places. Each coordinate will only be 0.00010, 0.00010 from previous coordinate.

- ✓ {{“-3.00115, 37.13495”,…, “-3.06142, 37.31296”}, true}
- ✓ {{“-3.28395, 37.52273”,…, “-3.07632, 37.35410”}, true}

3. **setCorrectionNotice(str) : bool**
   Invalid input: any number such as int values and double values. It will also be invalid if nothing is entered.
   - × {-78, false}
   - × {24, false}
   - × {8.0, false}
   - × {“ ”, false} (“ ” represent an empty/nothing entered)

   Valid input: Any string will be valid. (errors in spelling or grammar will be valid so be careful)

   - ✓ {“There are 5 rebels nearby”, true}
   - ✓ {“Trail 5 is unsafe due to falling rocks”, true}
   - ✓ {“Weather is unsafe to hike today”, true}

Integration Tests

Testing the relationship between components within the system for each test

In our test plan, we want our integration tests to cover the functions of the basic user class, trekker in this case. This is the most important part of the system: being able to retrieve the data such as the weather, or the trail condition - it is the whole reason this system even exist. With that said, it is crucial that we test the links between the user class and the information holding classes like Emergency and Event.

1. **getTrailCondition(Trail) : str**
   Invalid input: in this case, what is mainly being entered and the most crucial part would be the trailNum as it will create the report. So, like the unit test, its input would be any number less than 1, any number larger than the total amount of trails (in this case it would be a number larger than 12 or null input). However, since it is returning a string, the “false” equivalent would be a null string.
   - × {-5, “”}
   - × {0, “”}
   - × {“”, “”}
   - × {93, “”}

   Valid input: any number between 1 and the max number of trails (at this time it is 12, but could increase). However, since it is returning a string, the “true” equivalent would be the trail report in the form of a string.

   - ✓ {6, “Trail 6 is open for trekking”}
   - ✓ {10, “Trail 10 is closed for trekking”}

- ✓ {(number between 1 and max number of trails), “Trail (number) is open (or closed) for trekking”}

2. **getEventInfo(Event) : str**

   Invalid input: in this case the main component that will vary is the event name since the user would enter the name of an event, they want the info for, and the function will output a string which contains things like the location and date/time of said event. So invalid input would be a null name, anything other than a string such as a number, a string that doesn’t contains letters or simply something that isn’t an event and if it’s invalid it would return a null string.

   - × {“”, “”}
   - × {503, “”}
   - × {“+-..!#$”, “”}
   - × {“milk”, “”}

   Valid input: any valid event name, more specifically the name of an ongoing or future event that will happen in the park, and it would then return a full event report in the form of a string.

   - ✓ {“Summer BBQ”, “Summer BBQ will take place on August 4th (12:15pm) at Mandara Hut”}
   - ✓ {“Jeanne’s Yoga Class”, “Jeanne’s Yoga Class will take place on August 20th (08:15am) at Marangu Gate”}
   - ✓ {“(any valid event name)”, “(Event name) will take place on (date) (time) at (location)”}

3. **getWeatherReport(Weather) : str**

   Invalid input: in this case, the wrong information or wrong order. It should be weather condition first then temperature; if these are inputted wrong, then it will be invalid.

   - × {“Tuesday, 95 °F, sunny”, “”}
   - × {“thursday, sun, 94 °F”), “”}

   Valid input: a valid weather condition (rainy, sunny, cloudy, snowy), a valid day of the week and a valid temperature within the park. It would then return a full weather report in the form of a string.

   - ✓ {“sunny, Monday, 95 °F”, “Monday will be sunny with a temperature of 95 °F”}
   - ✓ {“snowy, Friday, 32 °F”, “Friday will be snowy with a temperature of 32 °F”}
   - ✓ {“(any valid weather condition), (temperature)”, “(day) will be (condition) with a temperature of (average temperature)”}

System Tests

Testing the entire system

In our test plan, we want the system test to essentially be a collection of integration tests grouped together in which every single component is reached at least once. We want to make sure that after running this system test, we can be sure of the fact that each element in this system works to some degree (without seeing/writing the source code)

1. **editWeather(editAccess, Weather) : bool**
   editTrail(editAccess, Trail) : bool
   editEmergency(editAccess, Emergency) : bool
   editEvent(editAccess, Event) : bool
   viewCamera(Weather) : void

   Invalid input: There could be several invalid inputs when dealing with this large number of tests, but I think it is important to hit the main ones and perhaps the ones that could cause the most faults. First and foremost, if at any time editAccess is false, it is crucial that it returns false and allows no edits through. Another invalid input would be any case where you are trying to edit something like an event or a trail and enter a null string. Lastly, it would be hard to test since it returns void. But say that a user tries to view the camera and isOnline is false, it should show no video.

   × {true, " ", false} – leaving a blank field in the weather class
   {false, 12, false} – Correct trail number but no edit access
   {true, "21f3fea89", false} – Invalid form of location for the emergency
   {false, "Trail 8 Cookout", false} – Correct event name but no edit access
   {false, (no camera view)} – isOnline is false so the camera can't show anything

Any combinations of these failure would then result in a failure of the system test, more specifically these are all examples of how to incorrectly use the system and how we expect it to react to such errors.

Valid input: In terms of valid input for our particular system testing: it is necessary for editAccess to be true so that user can successfully edit the system's information. Furthermore, trailNum's input requires an actual trail number so that the input is valid. Additionally, valid input when testing to view the camera requires isOnline to be true so that camera can be accessed.

✓ {true, {{45, 47,…, 54, 51}, {53, 53,…, 48, 47},…, {48, 52,…, 51, 51}}, true } – Correct temperature format of 2D-array with edit access
{true, 4, true} – Correct trail number with edit access
{true, "-3.08138, 37.38907", true} – Valid form of location for the emergency with edit access
{true, "Trail 8 Cookout", true} – Correct event name with edit access

{true, (camera view)} – isOnline is true so the live camera feed is shown

2. **getWeatherReport(Weather) : str**
   getTrailCondition(Trail) : str
   editEvent(editAccess, Event) : bool
   viewCamera(Weather) : void
   checkEmergency(Emergency) : str

   Invalid input: For the most part, this will be very similar to the previous system test. Things like editAccess being false at any point should return false for the edit functions. In terms of the get functions, formatting issues such as leaving fields null or inputting numbers for strings and vice versa. And once again, if isOnline is false then it will show no live feed of the camera.

   × {“ ”, “”} – leaving a blank field in the weather class, results in an empty string being returned
   {22, “”} – Nonexistent trail number inputted results in an empty string being returned
   {true, “4th of July Fireworks”, false} – Valid editAccess but invalid event entry results in a false Boolean being returned
   {false, (no camera view)} – isOnline is false so the camera can’t show anything
   {“-2.9562, 37.4988”, “”} – Not precise enough location (invalid), returns empty string

   Valid input: Again, just like the last system test, valid input includes correct formatting of class attributes such as having a trail number be in the range of valid numbers, having real events, no null inputs and finally, having the camera’s isOnline be true.

   ✓ {“cloudy, Thursday, 53 °F”, “Thursday will be cloudy with a temperature of 53 °F”}
   {8, “Trail 8 is open for trekking”} – Valid trail number that is between 1 and number of total trails
   {true, “Summer BBQ”, true} – Valid editAccess with true event being accessed
   {true, (camera view)} – isOnline is true so the live camera feed is shown
   {“-2.95615, 37.49880”, “Current rebel activity at this location”} – Valid form of GPS coordinate that then returns emergency notification as string