Kili Trekker System Test Plan

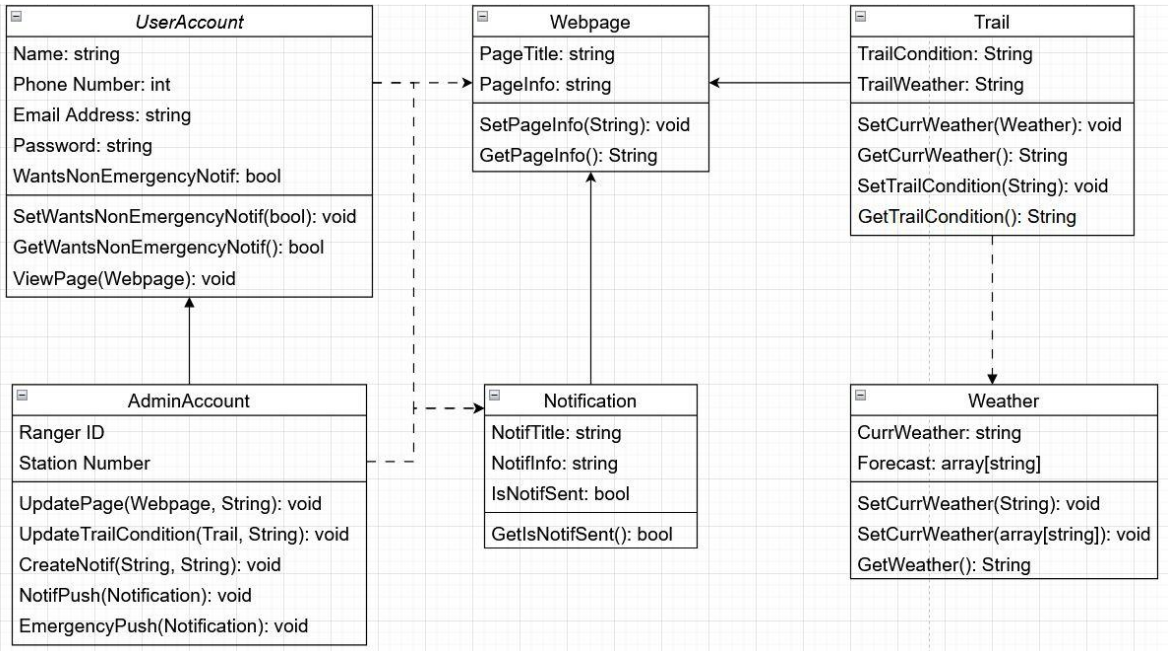
Prepared by: Matteo Gristina, Ryan Jaber, Neel Raj

March 23, 2023

Table Of Contents

1 Software Design Specification

1.1 UML Diagram



1.2 Diagram and Class Relationship Explanation

The UML diagram has been almost completely overhauled with testing in mind. Functions have been added to clearly define features of the system and integrate with each other. The Kili Trekker System is made up of a strong core of classes, the account classes, the webpage classes and the weather class. There are two account classes, one for normal users, which includes basic identification information, and the admin account. AdminAccount has an “is a” relationship with the UserAccount class, and inherits basic information from the UserAccount class, but includes functions that are only accessible to admins, which help them update and maintain information in the system. Next, the webpage classes include the base webpage class, the notification class, and the trail class. The Webpage class is very barebones, but gets elaborated/specialized with the Notification and Trail classes, which have an “is a” relationship with the Webpage class. The notification is a special type of webpage, shown to the user in special circumstances, and sent by an administrator. To synergize with this behavior, a bool is added to show the status of the notification. As for the trail class, two important characteristics are added on top of inherited attributes. The weather, like sunny, cloudy, raining, etc. and the condition of the trail, good, fine, bad, flooded, closed, etc. The Trail class has a “uses” relationship with the Weather class in order to update the weather status, and display accurate information.

1.3 Attribute + Function Explanation

This section aims to clear up some non-obvious attributes and functions in the UML Diagram. First, in the UserAccount class, is the **ViewPage(Webpage)** function. The ViewPage(Webpage) function will show the user the webpage passed in the argument, creating a level of abstraction between the user and the system. Also, to bring a quality of life change, the user will have the ability to choose their preference on non emergency notifications. This will be achieved with the **Set** and **GetWantsNonEmergencyNotif** functions.

In the trekker system, approved users like admins and rangers must be able to write to and update information hosted in the system. Therefore, in the AdminAccount class, there are a number of important functions, UpdatePage, UpdateTrailCondition, CrateNotif, NotifPush, and EmergencyPush. **UpdatePage(Webpage, String)** updates the information section of the passed webpage, letting admins create detailed web pages with relevant information for the user. **UpdateTrailCondition(Trail, String)** allows a ranger/admin to update the condition of a trail in case its condition changes, like flooding, or closing the trail. **CreateNotif** passes a string which gets set as the notification’s NotifTItle, letting admins create notifications based on live scenarios. **NotifPush** and **EmergencyPush** both pass a Notification object, the difference between the two functions is one pushes the Notification as an emergency, reserved for notifications that need to be seen urgently by users and contain very important information.

The Weather class has an overloaded **SetCurrWeather** function. One where you can pass a String, and another where you can pass an array at a certain index. These end up both passing strings, but it is overloaded to explain the intent behind the Forecast attribute. Lastly, in the Trail class, the **SetCurrWeather** function utilizes the Weather class as an object to manipulate the TrailWeather attribute in the Trail class.

2 Verification Test Plan

2.1 Unit Tests

2.1.1 UserAccount: WantsNonEmergencyNotif

Test Set 1: A part of the Kili Trekker system is to send notifications to users. To help admins and rangers, there are two types of notifications, emergency and non-emergency. Some users may not prefer to have non-notifications active, so a feature of the system is to let users toggle their preference. Test Set 1 tests this feature, manipulating the WantsNonEmergencyNotif attribute within the UserAccount class.

```
#Test Case 1#
input:
new UserAccount user1
user1.SetWantsNonEmergencyNotif(true)

expected output:
user1.GetWantsNonEmergencyNotif()
with return value “true”

#Test Case 2#
input:
new UserAccount user1
user1.SetWantsNonEmergencyNotif(false)

expected output:
user1.GetWantsNonEmergencyNotif()
with return value “false”
```

These test cases cover the “non-emergency notification preference” feature fully because there are only two possible things that the bool WantsNonEmergencyNotif can be. No matter what the value of the bool is before SetWantsNonEmergencyNotif is run, the output is expected to be the bool passed by the set method. If the output gained from testing does not match the expected outputs in the test cases, then the feature has been implemented incorrectly, and we know that the error lies somewhere in the UserAccount class, because no other classes are involved in the test.

2.1.2 Webpage: PageInfo

Test Set 2: A main part of the Kili Trekker System is the Webpage structure, as well as the editing page for info is dramatically important. The manipulated data is PageInfo in the webpage class. The webpage component allows for Rangers and staff to input the necessary data that needs to be displayed on the webpage. The system's way of editing the page info is to pass a string as the argument through the SetPageInfo command in the command prompt of the operating system.

```
#Test Case 1#
input:  SetPageInfo(“null”)

expected output:
**null**

#Test Case 2#
input:  SetPageInfo(“R”)

expected output:
R

#Test Case 3#
input:
SetPageInfo(“**Full Page of Special Characters**”)
**No ASCII and non unicode**

expected output:
**Full Page of Special Characters**
```

These test cases fully covers the targeted feature because we test edge cases for a string through test case #2. If a char can work, then two characters can likely work, hence all the characters will. In addition, test case #1 is meant to experiment with the input of null and when considering all possible inputs for a string, there is only one with null. The expected output of the webpage should be empty or null to indicate the webpage compilation correctness. Test case #3 is meant to experiment with the character capabilities that the ranger or editor wants to incorporate. Using the full page of special characters tests the boundaries and edges of all the webpage space, as well as distinction of characters that can and may be used.

2.2 Integration Tests

2.2.1 Trail: TrailCondition

Test Set 3: A part of the Kili Trekker system is to let admins write and publish information through the system for the user to view. To help admins and rangers, the AdminAccount class has functions directly intended for that purpose. The UpdateTrailCondition function passes a Trail object and a string to manipulate the TrailCondition attribute of the passed Trail object. This lets an admin or ranger manually update trail conditions without needing to directly change exact attributes in all of the trail objects.

```
#Test Case 1#
input:
new AdminAccount admin1
new Website testTrail
admin1.UpdateTrailCondition(testTrail, “closed”)

expected output:
ERROR with explanation: “Website object is not of type
Trail”

#Test Case 2#
input:
new AdminAccount admin1
new Notification testTrail
admin1.UpdateTrailCondition(testTrail, “closed”)

expected output:
ERROR with explanation: “Notification object is not of
type Trail”

#Test Case 3#
input:
new AdminAccount admin1
new Trail testTrail
admin1.UpdateTrailCondition(testTrail, “closed”)
```

```
expected output:
testTrail.GetTrailCondition()
with return value “closed”

#Test Case 4#
input:
new AdminAccount admin1
new Trail testTrail
admin1.UpdateTrailCondition(testTrail, null)

expected output:
testTrail.GetTrailCondition()
with return value null

#Test Case 5#
input:
new AdminAccount admin1
new Trail testTrail
admin1.UpdateTrailCondition(testTrail, “c”)

expected output:
testTrail.GetTrailCondition()
with return value “c” (string)
```

These test cases cover the “set trail condition” feature fully because there is only one valid Website class that has the attribute TrailCondition, the Trail class. Website is the parent class, and Notification is a child of Website, but both do not include the correct attribute. In addition, the string portion of the input is tested with values of null, c, and closed, which are cases very near to the partition boundaries of the String datatype. If the output gained from testing does not match the expected outputs in the test cases, then the feature has been implemented incorrectly, and we have located an error that has resulted in a failure.

2.2.2 Trail: CurrWeather

Test Set 4: A part of the Kili Trekker system is to eventually have the system automatically source and update its weather information. The trail class contains the SetCurrWeather which passes a weather object. The weather object has a CurrWeather attribute, which is passed into the method in order to set the Trail's CurrWeather attribute. The function we are testing is the SetCurrWeather in the Trail class, not the weather class.

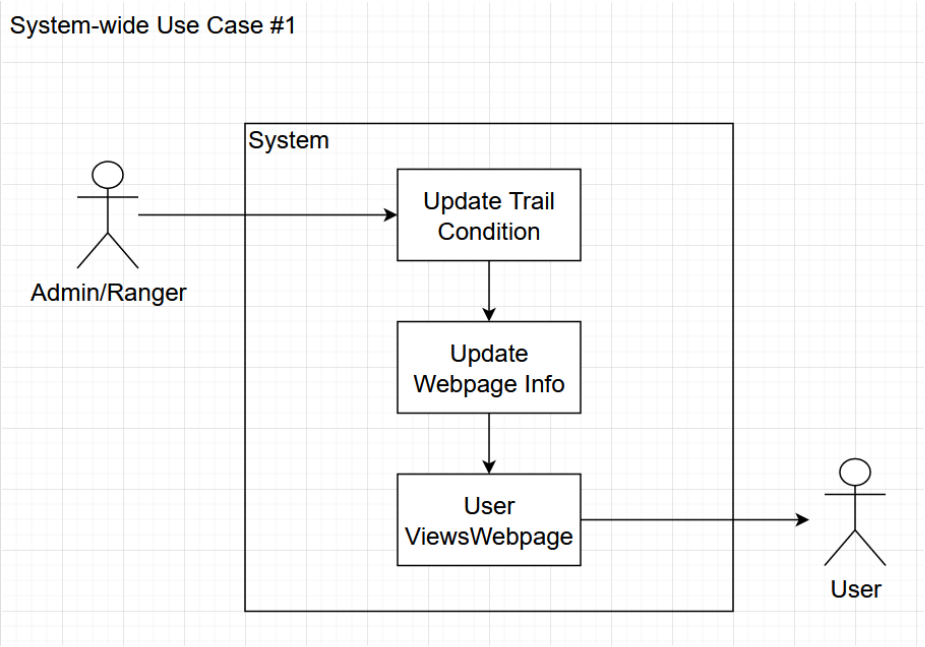
```
#Test Case 1#
new Weather todaysWeather
new Trail testTrail
todaysWeather.SetCurrWeather("Sunny")

input:
testTrail.SetCurrWeather(todaysWeather)

expected output:
testTrail.GetCurrWeather()
returns "Sunny"

#Test Case 2#
new [ANY CLASS NOT WEATHER] todaysWeather
new Trail testTrail

input:
testTrail.SetCurrWeather(todaysWeather)

expected output:
Error, [CLASS NOT WEATHER] does not match object type
Weather.
```

These test cases fully cover the targeted feature of updating Trail CurrWeather, because it ensures that the class needs to be the appropriate Weather class. The input space for Trail's SetCurrWeather function is all classes instantiated as objects, but the only one that can function as intended is the Weather class, because an attribute from the weather class is directly passed to the Trail Class. If the input object does not have the same type as Weather, the system cannot properly update values, so when automating the weather updating system, this needs to be considered.

2.3 System Tests

2.3.1 System Test 1: Trail Condition

System Use Case 1: The Kili Trekker System allows approved rangers to update and maintain relevant information about various things, but mainly trails, which will then be displayed to the user. Users cannot change the information provided to them, and instead, can only view. This updating of information is the feature that is targeted in this first system test. The scenario is that a trail needs to be closed, and a ranger needs to update and make this information consistent in the system. It will ultimately involve the AdminAccount, UserAccount, Trail, and Webpage classes. The ranger will update the condition of the now closed trail, but this attribute is not directly viewable to the user. Then the admin will update the trail's webpage info, saying when the condition was last monitored or changed, along with any relevant information regarding details of trail closure. The last step is for the user to view these changes on the website, which confirms that the information has in fact changed and is consistent across the board. This use case is summarized in the following use case diagram:




```
#Test Case 1#
new UserAccount user1
new AdminAccount admin1
new Trail testTrail
```

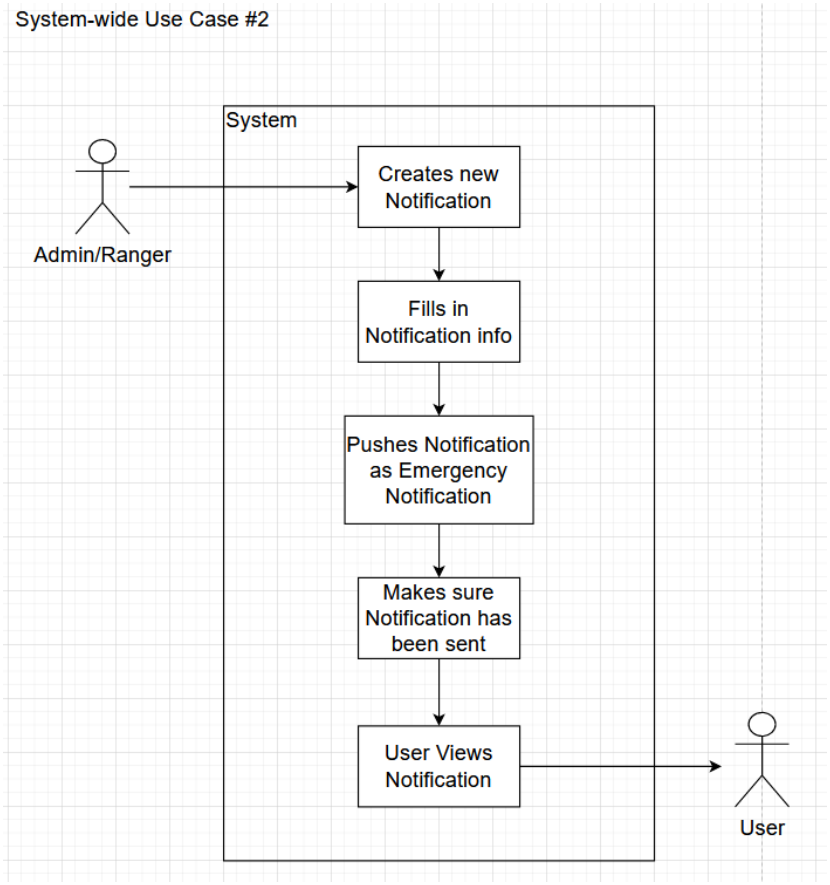
```
admin1.UpdateTrailCondition(testTrail, “trail is
open”)

input:
admin1.UpdateTrailCondition(testTrail, “closed”)
testTrail.SetPageInfo("trail is now closed until
further notice")
user1.ViewPage(testTrail)

expected output:
User views “testTrail: trail is now closed until
further notice”

expected output 2:
testTrail.GetTrailCondition()
with return value “closed”


#Test Case 2#
new UserAccount user1
new AdminAccount admin1
new Trail testTrail

admin1.UpdateTrailCondition(testTrail, “trail is
open”)

input:
user1.UpdateTrailCondition(testTrail, “closed”)
testTrail.SetPageInfo("trail is now closed until
further notice")
user1.ViewPage(testTrail)

expected output:
User views “testTrail: trail is now closed until
further notice”
```

```
expected output 2:
Error, user1 does not have permission to use
UpdateTrailCondition.
testTrail.GetTrailCondition()
with return value “trail is open”
```

The TrailCondition feature has been covered in depth in section 2.2.1, but this test builds upon that feature, adding interaction from the admin and user this time. The second test case shows a user, who is unauthorized to use the UpdateTrailCondition. The expected response from the system is that it recognizes this lack of permission, and as a result the trail condition is unchanged from what it was set to in the beginning. This fully tests the targeted feature because the system has been extensively tested with TrailCondition and strings, but has not encountered different UserAccounts trying to access certain commands. The test cases presented finally test the implementation of the permissions and access within the Kili Trekker System.

2.3.2 System Test 2: Emergency Notification

The Kili Trekker System allows for immense contact with the emergency notification procedure. The rangers that are given admin status may create and allocate emergency information pushes to users and other rangers. Users cannot make emergency notifications and will have to reach out to a ranger to have the admin(s) ranger(s) begin emergency notification procedures. An example scenario case is an Admin Ranger needs to push a new emergency notification out. To start, the Admin will be the defining factor of beginning the emergency procedure, by determining if what has been said or done is an emergency. Components that are needed for this system test are AdminAccount, Notification, and Webpage classes. The Admin Ranger decides to push the emergency notification, hence, users and other rangers will be notified and the emergency will be shown on the display. This use case is shown in the use case diagram steps.



#Test Case 1#
new UserAccount user1
new AdminAccount admin1
admin1.CreateNotif(“TestEmergency”, “This is a test”)

input:
EmergencyPush(TestEmergency)

expected output:
IsNotifSent()
with return value “true”

expected output 2:
user1.ViewPage(TestEmergency)
Users on the system receive the emergency notification with title “TestEmergency” and description “This is a test”

#Test Case 2#
new UserAccount user1
new AdminAccount admin1
admin1.CreateNotif(null, “This is a test”)

input:
EmergencyPush(null)

expected output:
Error, null would be passed as the title of the NOtification object, so the object would not be able to be referenced. The system should not allow admins to make null objects.

```
#Test Case 3#
new UserAccount user1
new AdminAccount admin1
admin1.CreateNotif(“TestEmergency”, “This is a test”)

input:
EmergencyPush(TestEmergency1)

expected output:
IsNotifSent()
with return value “false”
Error, no Notification called TestEmergency1

#Test Case 4#
new UserAccount user1
new AdminAccount admin1
new Webpage TestEmergency

input:
EmergencyPush(TestEmergency)

expected output:
IsNotifSent()
with return value “false”
Error, no Notification called TestEmergency1. Webpage
is parent class, so no notification object titles
TestEmergency exists.
```

These test cases present various conditions for success and failure when pushing an emergency notification. This covers issues with the input string(s), the object types, and the specific notification passed to EmergencyPush. This fully targets the intended feature becuase it tests different inputs at each stage of the push notification process. The input for starting the emergency notification procedure begins with the EmergencyPush command being filled out with information. Once the emergency notification is ready to deploy, the expected output should have a boolean value letting know if the notification was pushed successfully. Also, make sure to check a test ranger phone for the emergency notification as well.