Software Design Specification - Kili Trekker System

Prepared by Erica Lee, Jennifer Giovengo, and Bryce Garrod

April 22, 2023

System Description

The Kili Trekker Tracker System is a sophisticated software developed for the Tanzanian government to enhance tourism in the breathtaking Mount Kilimanjaro region on the Tanzanian-Kenyan border in Africa. Designed primarily for trekkers visiting the Kilimanjaro National Park, this system offers many features that allow hikers to access vital information and track their location while exploring the area. As well as providing real-time information about the twelve trails on Mount Kilimanjaro, the software also helps trekkers organize authorized trail guide companies to accompany groups to the summit with various paths, and guides those who signed up with guides to hike around the area. In addition to these features, the Kili Trekker Tracker System provides an extensive range of services to users, including tips, information, and disclaimers for those taking the solo route. Local events around the area, weather forecasts, and trail conditions can also be accessed. The weather tab even features a camera at the top of Mount Kilimanjaro to offer a real-time view of the summit and trail conditions. In case of emergencies, the software promptly displays notices about mandatory evacuations, warnings, or threats due to Kenyan rebels, ensuring that trekkers are kept informed at all times. Authorized personnel such as park officers, park rangers, guides, and IT technicians can have full access to the system via two-factor authentication to perform various functions, such as updating information, maps, trails, and available times for guides.

Software Architecture Overview

Architectural diagram of all major components

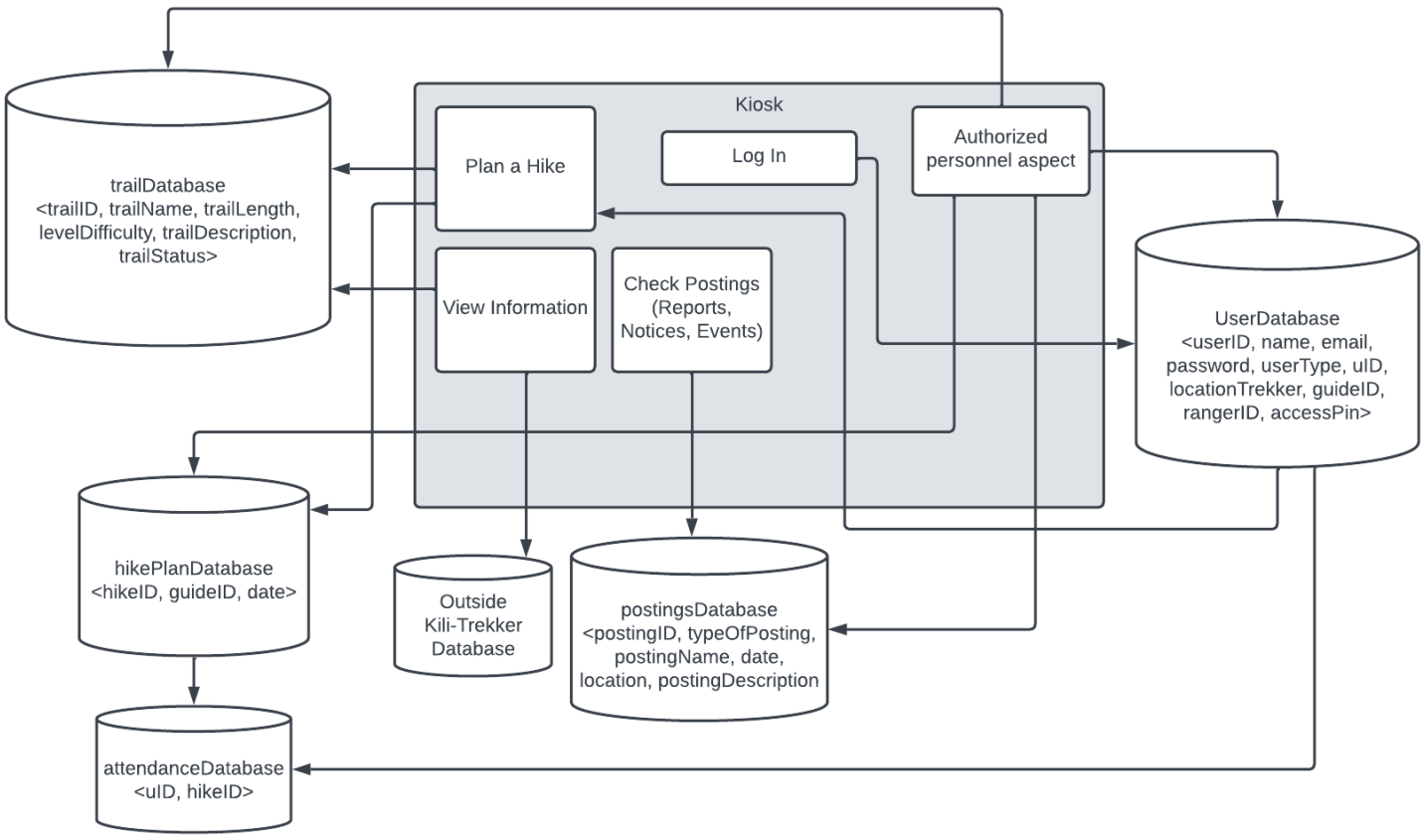

Figure 1: Software Architecture Diagram for the Kili Trekker Tracking System

UML Class Diagram

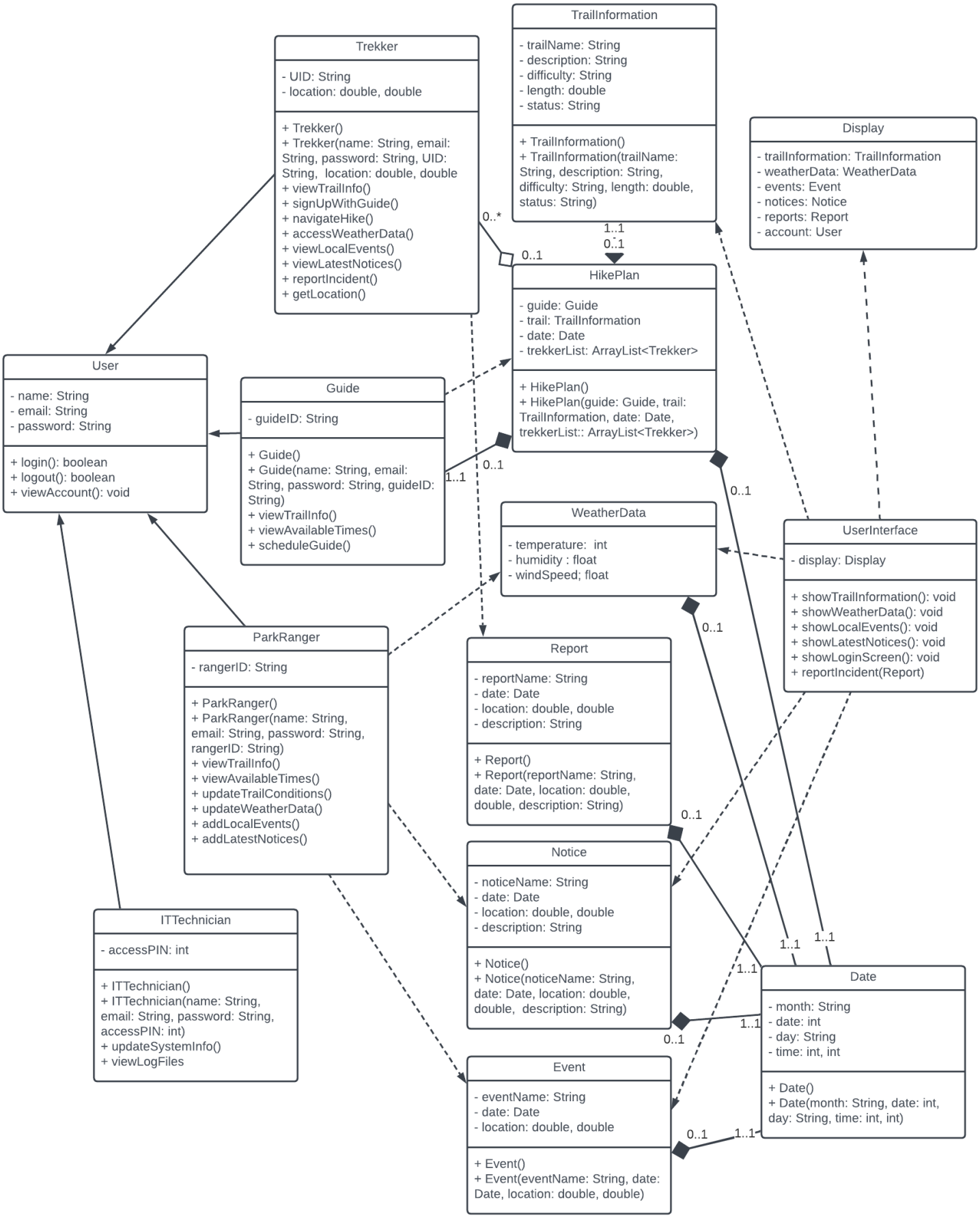

Figure 2: UML Diagram for the Kili Trekker Tracking System

Old Software Architecture Diagram Description

(Figure is no longer attached due to new amendments. Refer to Figure 1 for new Software Architecture Diagram.) Any authorized personnel has the ability to access the user database to manage and access user information. This could be for recovering user accounts that have been lost by the user or to fix something about a user's account that needs to be altered. Authorized personnel can also get access to and modify any information regarding trails, available guides, notices, and general information. Alteration of these databases could be necessary if a trail has been destroyed or is inaccessible, a guide quits working at the mountain or a new guide is hired, a new notice needs to be made for an event or a danger, or adding any piece of general information that is needed by trekkers or deleting obsolete general information. As a user, there are three main types of functions that a user would want to access: planning and making reservations for a hike, checking current park news, and viewing general information about the park. To access planning a hike, the user must log in and the information from the user database will be used to verify the login is valid. When planning a hike, the trail that will be taken and the guide who will be with the group are needed, so the plan a hike function is connected to both the trail and guide databases. For checking notices, all notices, events, and weather news are in the same database, and the check notices function is connected to the notices database. The way the database receives information is either by an authorized person manually altering something like creating a new event or by sourcing it from an outside database, like updating the weather from a weather station. Finally, the view information function accesses the information database, which consists of general park information that is sometimes updated by authorized personnel.

New Software Architecture Diagram Description

(Figure 1.) Any authorized personnel has the ability to access the user database to manage and access user information. This could be for recovering user accounts that have been lost by the user or to fix something about a user's account that needs to be altered. Authorized personnel can also get access to and modify any information regarding trails, hike plans, postings, or any general information. New hike plans should be listed daily with their corresponding attributes. The Attendance database is for logging all values of trekkers who sign up for the hike. It uses attributes of the trekker's uID and their hikeID, and we obtain this data from the hikePlan and user databases.
As a user, there are three main types of functions that a user would want to access: planning and making reservations for a hike, checking the latest postings, and viewing general information about the park. To access 'Plan a Hike', the user must log in and the information from the user database will be used to verify the login is valid. When planning a hike, hikePlan's databases hold information on what guides are available for the hike of an allotted time and designated trail. For checking the postings, all notices, events, and reports are found in this database. This database receives its information from authorized personnel who can add new items such as a new local event in the area or an action of parsing through listings to find older postings. The action 'View Information' refers to the information outside the Kili-Trekker Databases which is sourced from an outside database, like updating the weather from a weather station.

Description of classes, attributes, operations

(Figure 2.)
User class
The User class manages all users in the system including trekkers, guides, park rangers, and IT technicians. These listed users share a generalization relationship with the User class. Not all trekkers need to have an account, however, an account needs to be used for those who wish to sign up for a guided hike. Attributes of this class include name, email, and password which are then inherited by the four users. Using these attributes, users can perform actions such as logging in, logging out, and viewing their account details under viewAccount(). These operations act as getter functions.

Trekker class
The Trekker class encapsulates the actions and functions that a trekker can do. Even though the trekker may or may not have an account, the class allows them to view trail information, read the information on how to navigate their hike, and access weather data. Attributes of this class include user identification (UID) as a String value of a combination of letters and numbers as well as live tracking of their current location, stored as doubles for latitude and longitude, as long as they are logged on their mobile device in the Kilimanjaro vicinity. For all four users, we implemented their respective constructors, in this case, we have the Trekker() constructor which is an instance of the Trekker class that creates an object, Trekker. Trekker also inherits information from the User class, including the name, email, and password. The functionality of the Trekker class includes being able to view trail information, navigate hikes, access weather data, view local events, view the latest notices, and report incidents. However, to sign up with a guide, the trekker would need to have an account under the User class to book the time slot.

Guide class
The Guide class inherited from the user class This class, inheriting attributes of the User class, includes all the functions that a guide leader can do. As mentioned with the Trekker, we have the Guide() constructor, an instance of the Guide

class that creates an object, Guide. Because the trail guide leaders refer to the trails made available to be scheduled with, the Guide class depends on the TrailInformation class. An additional attribute of the Guide class that wasn't inherited from the User class is the guide leader's Guide ID, a String value of a combination of letters and numbers, which they use to log into the database to access trail info, available times, and the guide schedule. Guide leaders can perform functions such as viewing trail information, and available times that they can schedule a guide with the tourists. They can also schedule appointment slots for the company-led guides themselves based on the 12 trails that Kilimanjaro's National Park has.

ParkRanger class
The ParkRanger class– like the Trekker and Guide class– takes in the inherited attributes of User: name, email, and password, along with its own attribute: rangerID used for authentication. An instance of ParkRanger is also used as an Object. The Park Ranger class consists of all the actions and functions a park ranger can do: viewing trail information, viewing available times, updating trail conditions, updating weather data, adding local events, and adding the latest notices. In terms of hierarchy, the park ranger performs most of the updating of information, while the Trekker and Guide classes have limited operations. It also is dependent on the latestNotices class and localEvents class.

ITTechnician class.
Like the other three inherited classes from the User class, the ITTechnician also acts the same where a constructor of the ITTechnician class is created as an instance. The ITTechnician class inherits the User class's attributes: name, email, and password, and includes its own attribute: accessPIN which is used for updating the system information and for viewing log files to access the User database.

TrailInformation class
The TrailInformation class manages all the details that make up each of Kilimanjaro's 12 hiking trails. This includes the trail names, their respective descriptions, the level of difficulty of each trail, and a double data type describing the length of each trail (in kilometers). There is an instance of the TrailInformation class of type TrailInformation that acts as an object and sets values for any existing object properties like the name, description, difficulty, and length. The HikePlan is dependent on the TrailInformation because, without the trail attributes, there is no information on anything to hike on.

HikePlan class
This class encapsulates all the information needed for each planned hike. Each HikePlan has a Guide, TrailInformation, and a date associated with it. Attributes include the guide as an instance of Guide, the trail as an instance of TrailInformation, a date as an instance of Date, and the ArrayList of trekkers (the Trekker objects) signed up for the hike. Operations include HikePlan() as an empty parameter constructor, HikePlan(guide: Guide, trail: TrailInformation, date: Date, trekkerList: ArrayList<Trekker>) WeatherData class WeatherData class provides temperature information, humidity, and wind speed. UserInterface relies on the WeatherData class to display information. WeatherData is extracted from an outside database that provides the class with values for variables: temperature (int), humidity (float), and windSpeed (float).

Report class
The Report class consists of the elements that make up a report, used if a trekker wants to report an incident. The report name and the user-provided description of the report are two attributes using the String data type. The location is provided as two doubles to represent latitude and longitude. The Report class is associated with the Date class meaning it accesses the Date class to use the current month, date, day, and time for the reported incident.

Notice class
The Notice class has the attributes: noticeName, date, location, and description. The LatestNotices class is used by the user-Interface class meaning that LatestNotices is associated with. It is also associated with the ParkRanger class. LatestNotices provides a string noticeName, along with a dependency on the Date class. This class also provides a location.

Event class
The Event class includes the components that make up an event announcement, including the attributes, eventName (data type: String), date (attribute of type Date that holds month, date, day, and time), and location (data type double, double (representing the latitude and longitude). Event() is a constructor operation that creates an object of the Event class without any parameters. Event (eventName: String, date: Date, location: double, double) is instantiated as an object with parameters: eventName, date, and location.

Date class
The Date class includes the month (String), date (int), day of the week (String), and time (int, int). Other classes such as HikePlan, WeatherData, Report, Notice, and Event compose the Date class for their date attribute. Date also has its own Object instantiated which takes in the parameters of month, date, day, and time.

Display class
The Display class has the UserInterface class dependent on it. Attributes include trailInformation with the data type, TrailInformation; weatherData with the data type, WeatherData; events with the dataType, Event; notices with the data type, Notice; reports with the datatype, Report, and account with the data type, User. These all include the instantiated objects from all their respective classes, which allow them to access the parameters to display the actions under UserInterface.

UserInterface class
The UserInterface class uses the Display class to show information given the specific method. Attributes are showTrailInformation, showWeatherData, showLocalEvents, showLatestNotices, showLoginScreen, which are all getter functions, which return void. reportIncident is a setter function that takes in the attributes of the Report object.

Development Plan and Timeline

The Kili Trekker Tracking System will have three main phases of development: user types development, user-created objects development, and user interface development, partitioned among team members Jen, Erica, and Bryce, respectively The user types development stage will consist of making the Trekker, Guide, ParkRanger, and ITTechnician classes, the instances of which will store the account information for a certain kind of user. During the development of these classes, making sure each class has the correct access to the features of the program is important to make sure that each type of account can be used correctly. To make sure that nobody gets access to an account type they should not have access to, the security of the program should be made robust with whatever encryption techniques the cybersecurity person or group sees fit. After the different types of users are created, the objects which they can create and access the information of will be developed. Through the user interface, any class should be able to access the weather data, which will be sourced from a weather API. Trekkers should be able to create reports, and those reports should be able to be viewed by users that are park rangers. Guides should be able to create hike plans and these should be available for all to access. Park rangers should be able to create notices and events, and those notices and events should be available for all to access. Finally, the last phase is to create the user interface and allow each type of user to create the objects they should be able to create and access the objects they should be able to access in an easy-to-understand format.

Unit, Functional/Integration, and System Testing

Unit tests for User Class

Test Case ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Status (Pass/Fail)
TC_LOGIN_001	Check if the user can log in to their account with the correct credentials: email and password.	Enter a valid email and a valid password.	The user already has a valid account.	1. Enter a valid email. 2. Enter a valid password. 3. Click the login button.	user123@mail.com P4s5w0rd000!	The boolean value should return true. Successful login.	The user is redirected back to the page they were previously at prior to logging in. The user remains logged in.	The boolean value returns true. Successful login.	PASS
TC_LOGIN_02	Check if the user can log in to their account with the correct credentials: email and password.	Enter a valid email and an invalid password.	The user already has a valid account.	1. Enter a valid email. 2. Enter an invalid password. 3. Click the login button.	user123@mail.com password000!	The boolean value should return false. The message should display: "The email and password you entered do not match. Please try again."	The user stays on the login page. The user is not logged in due to insufficient credentials.	The boolean value returns false. The message is displayed: "The email and password you have entered do not match. Please try again."	PASS
TC_LOGIN_03	Check if the user can log in to their account with the correct credentials: email and password.	Enter an invalid email and a valid password.	The user already has a valid account.	1. Enter an invalid email. 2. Enter a valid password. 3. Click the login button.	user234@mail.com P4s5w0rd000!	The boolean value should return false. The message should display: "The email and password you entered do not match. Please try again."	The user stays on the login page. The user is not logged in due to insufficient credentials.	The boolean value returns false. The message is displayed: "The email and password you have entered do not match. Please try again."	PASS
TC_LOGIN_04	Check if the user can log in to their account with the correct credentials: email and password.	Enter an invalid email and an invalid password.	The user already has a valid account.	1. Enter an invalid email. 2. Enter an invalid password. 3. Click the login button.	user234@mail.com password000!	The boolean value should return false. The message should display: "The email and password you entered do not match. Please try again."	The user stays on the login page. The user is not logged in due to insufficient credentials.	The boolean value returns false. The message is displayed: "The email and password you entered do not match. Please try again."	PASS

Figure 3: Unit Test 1: User Class LOGIN

Test Case ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Status (Pass/Fail)
TC_VIEWACCT_001	Check if the user can view account information given that they are already logged in.	Select view account details while logged in. They verify that they are accessing the right account by reentering the password.	The user already has a valid account.	1. Go to settings. 2. Select view account details	P4s5w0rd000!	Users should be able to access their user account details	The user is able to view a page with their account details.	User is able to access their user account details. This includes their name, email, and password.	PASS
TC_VIEWACCT_002	Check if the user can view account information given that they are already logged in.	Select view account details while logged in. They verify that they are accessing the right account by reentering the password.	The user already has a valid account.	1. Go to settings. 2. Select view account details 3. Reenter invalid password	password000!	Users should not be able to access their user account details. A message should prompt them that their password is incorrect.	The user stays on the pop-up with the prompt to reenter their password due to insufficient credentials.	User is not able to access their user account details. A message tells them that their password is incorrect.	PASS
TC_VIEWACCT_003	Check if the user can view account information given that they are already logged in.	Select view account details while viewing as a guest.	The user already has a valid account.	1. Go to settings. 2. Select view account details	N/A	User should not be able to access their user account details. A message should prompt them to log in in order to view account details.	The user stays on the Settings page with a pop-up asking that they log in. The user is not logged in, so they cannot access their account information.	User is not able to access their user account details. A message prompts them to log in in order to view account details.	PASS

Figure 4: Unit Test 2: User Class VIEWACCT

Functional/Integration tests for Trekker Class

Test Case ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Status (Pass/Fail)
TC_CREATETREKKER_001	Verify that the Trekker object is created with valid inputs for UID, and location values (latitude and longitude). Retrieve the values of name, email, password, UID, and location values (latitude and longitude) and ensure that they match the given information	Create a Trekker object from the attributes of the User class with the usage of valid input parameters: UID and location	A user must exist to inherit the Trekker attributes.	1. Create a new Trekker object adding the UID and the location (latitude, longitude). 2. Retrieve the values of name, email, password, UID, and location values (latitude and longitude)	T942033392 -3.077295, 37.361887	John Doe jdoe@mail.com P4s5w0rd000! T942033392 -3.077295, 37.361887	A Trekker object is created with attributes from the User class inherited. Attributes of the Trekker object now include all values.	John Doe jdoe@mail.com P4s5w0rd000! T942033392 -3.077295, 37.361887	PASS
TC_CREATETREKKER_002	Verify that the Trekker object is created with valid inputs for UID, and location values (latitude and longitude). Retrieve the values of name, email, password, UID, and location values (latitude and longitude) and ensure that they match the given information	Create a Trekker object from the attributes of the User class with the usage of invalid input parameters: UID and location (eg. Negative UID, blank latitude, and invalid longitude	A user must exist to inherit the Trekker attributes.	1. Create a new Trekker object adding invalid values for UID and the location (latitude, longitude). 2. Retrieve the values of name, email, password, UID, and location values (latitude and longitude)	-3R4433392 , longitude	A message should display "The inputs you entered were invalid"	A Trekker object is not created with attributes from the User class inherited because attributes of the Trekker object do not include all values.	A message displays "The inputs you entered were invalid"	PASS

Figure 5: Functional/Integration Test 1: Trekker Class CREATETREKKER

Test Case ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Status (Pass/Fail)
TC_SIGNUPWITHGUIDE_001	Check if the user can sign up with a guide using the method signUpWithGuide( ). Make sure the user can access that page.	The user attempts to sign up with a guide while logged in.	The user already has a valid account.	1. Navigate to the page in the main menu to sign up with a guide. 2. Select a guide leader associated with one of the given trails and time slots. 3. Confirm	N/A	The user should be able to access a page to sign up with a guide.	User is directed to a page allowing them to sign up with a guide.	The user is able to access a page to sign up with a guide.	PASS
TC_SIGNUPWITHGUIDE_002	Check if the user can sign up with a guide using the method signUpWithGuide( ). Make sure the user can access that page.	The user attempts to sign up with a guide while viewing as a guest.	The user already has a valid account.	1. Navigate to the page in the main menu to sign up with a guide. 2. User enters valid information for login in order to continue	N/A	The user should not be able to access a page to sign up with a guide. Instead, a pop-up should appear, prompting the user to login in order to continue.	User is directed to a login page so they can enter their credentials in order for them to sign up with a guide.	The user is not able to access a page to sign up with a guide. Instead, a pop-up appears, prompting the user to login in order to continue.	PASS

Figure 6: Functional/Integration Test 2: Trekker Class SIGNUPWITHGUIDE

System tests for ParkRanger and UserInterface Class

Test Case ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Status (Pass/Fail)
TC_PARKRANGER_001	Check if the park ranger can perform all given methods in the class. The TrailInformation() class reads new updates based on the park ranger's input.	The park ranger attempts to update trail conditions by adding new information including trail name, trail description, trail difficulty, length, and status.	The park ranger is logged into the system with their rangerID.	1. Enter the trail name, trail description, trail difficulty, length, and status. 2. Retrieve the values of the trail name, trail description, trail difficulty, length, and status from the TrailInformation object.	"Marangu Route" "The Marangu route is a favorite of local tour operators as it's the shortest route and of an easy ascent." "Beginner" 5.7 "Closed due to heavy rain and unsafe trail conditions."	The TrailInformation() object should update based on the data provided by the park ranger input. Retrieved values should be the updated values of the TrailInformation object.	The system now contains the latest information for the trail's object	The TrailInformation() object is updated based on the data provided by the park ranger input. Retrieved values are the updated values of the TrailInformation object.	PASS
TC_PARKRANGER_002	Check if the park ranger can perform all given methods in the class. The Events() class reads new updates based on the park ranger's input.	The park ranger adds local events by uploading new fliers including information about holidays, special events, and any third-party events unaffiliated with the park.	The park ranger is logged into the system with their rangerID.	1. Enter the PDF file of the flier into a separate portal specifically for authorized personnel. 2. Retrieve the string file from the parkRanger() to Events() class.	Community-Picnic-Flyer-2023.pdf	The Event() class should have the new PDF flier from the park ranger's input. Retrieved files provided by the park rangers should be updated sequentially based on date and time.	The park ranger's new flier is listed sequentially above the other fliers previously posted.	The Event() class has the new PDF flier from the park ranger's input. Retrieved files provided by the park rangers are updated sequentially based on date and time.	PASS

Figure 7: System Test 1: Park Ranger Class (Multiple Functions)

Test Case ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Status (Pass/Fail)
TC_USERINTERFACE_001	Check if the userInterface() class can perform all given methods in the class. The userInterface() class relies on the Display() class to access the attributes of specific classes of the system.	The TrailInformation() class contents from the userInterface() are displayed through the Display() class by calling the showTrailInformation() function.	The data from each of these classes are all in the system ready to be read.	1. Call the function showTrailInformation(). 2. Verify that the userInterface() class correctly uses the Display() class to access information from the TrailInformation class.	N/A	The UserInterface() class should use the Display() class to show TrailInformation() data. This data should include the trail name, description of the trail, difficulty and length, along with the status of the trail.	The data from the TrailInformation() class is read correctly into Display() class.	The userInterface() class uses the Display() class correctly to show trailInformation() contents. The trail name, description of the trail, difficulty and length, and the status of the trail are accurately displayed.	PASS
TC_USERINTERFACE_002	Check if the userInterface() class can perform all given methods in the class. The userInterface() class relies on the Display() class to access the attributes of specific classes of the system.	The Report() class contents from the userInterface() is displayed through the Display() class by calling the showReport() function.	The data from each of these classes are all in the system ready to be read.	1. Call the function showReport(). 2. Verify that the userInterface() class correctly uses the Display() class to access information from the Report() class.	N/A	The UserInterface() class should use the Display() class to show Report() data. The date, location, and description should be displayed.	The data from the Report() class is read correctly into Display() class.	The userInterface() class uses the Display() class correctly to show Report() contents. The date, location, and description are correctly displayed	PASS
TC_USERINTERFACE_003	Check if the userInterface() class can perform all given methods in the class. The userInterface() class relies on the Display() class to access the attributes of specific classes of the system.	The Weather() class contents from the userInterface() is displayed through the Display() class by calling the showWeather() function.	The data from each of these classes are all in the system ready to be read.	1. Call the function showWeather(). 2. Verify that the userInterface() class correctly uses the Display() class to access information from the Weather() class.	N/A	The UserInterface() class should use the Display() class to show Weather() data. This includes the temperature, humidity, and wind speed.	The data from the Weather() class is read correctly into Display() class.	The userInterface() class uses the Display() class correctly to show Weather() contents: temperature, humidity, and wind speed.	PASS

Figure 8: System Test 2: User Interface Class (Multiple Functions)

Since Assignment 2, we have added a status attribute to the TrailInformation class in the UML diagram which holds a String with the status of the given trail because the current attributes to TrailInformation do not need to be changed often. With

the status of the given trail, users are informed on if any of the trails in the park are closed out of safety or a threat.

Data Management Strategy

Databases

For our data management strategy, we decided to use a total of five SQL databases that include the records which are arranged by row.

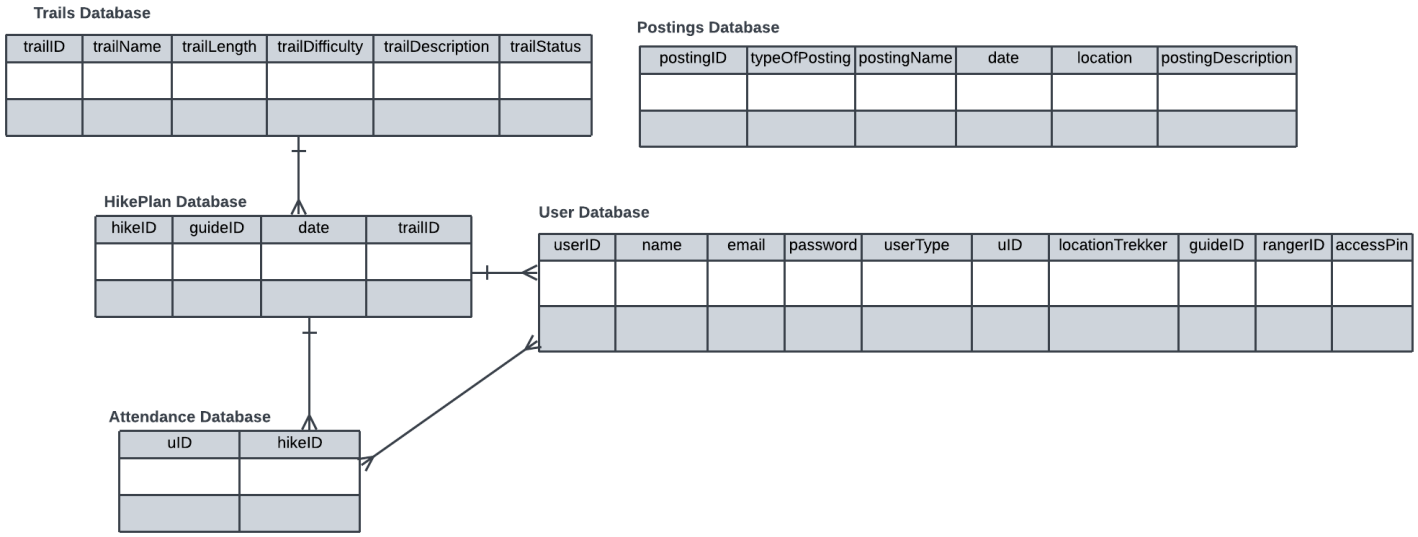

Figure 9: Data Management Strategy Diagram: Trails, Postings, HikePlan, User and Attendance Databases.

Trails Database
The Trails database is pretty straightforward; it holds data on the trails around Mount Kilimanjaro Park regarding trailID, trailName, trailLength, trailDifficulty, trailDescription, and trailStatus.

User Database
Each record in the User database represents a user (whether that be a trekker, guide, park ranger, or IT technician) and this includes their userID, name, email, and password as well as any additional attributes that are specific to their role. The trekker would also have additional attributes of uID and location, the guides would have their guideID, the park rangers would have their rangerID, and the IT technicians would have their access pin. (More of this will be explained in the Trade-Offs section).

HikePlan Database
The HikePlan database holds the record of every scheduled hike/guided tour with their respective guide. This is shown with the attributes of hikeID, guideID, date, and trailID. Each new hike plan will have its own entry so there should be new insertions of hike plans for each specific hike, differentiated by their attributes.

Attendance Database
The Attendance database holds the attendance of trekkers who attended a specific hike plan. Their uID (unique for every trekker) and their relative hikeID are the attributes of this database.

Postings Database
Postings Database consists of the postingID, typeOfPosting (this differentiates the postings of either events, notices, or reports), postingName, date, and location. There is no relationship between any of these databases with Postings.

Cardinality of the Five Databases

-The Trails Database and HikePlan Database share a one-to-many relationship. A trail is associated with many hike plans but only one hike plan can be associated with one trail.
- The HikePlan Database and Attendance Database have a one-to-many relationship because a hike can have multiple trekkers attending but a trekker can only attend one hike at a time.

- The HikePlan Database and the User Database share a one-to-many relationship. A guide can lead multiple hikes but a user can only attend one hike at a time.
- The User Database has a many-to-many relationship with the Attendance Database. A trekker can attend multiple hikes and a hike can have multiple trekkers attending.

Alternative Designs and Trade-Offs

There were a few decisions we had to make when we were handling our databases. Firstly was implementing inheritance in a database. For the User database, there are four types of users. We could have had four separate databases for each user with their specific attributes, but we decided on a single database that includes all of the attributes for the inherited users' data (trekker, guide, park ranger, and IT technician). This also includes an added value, which we have labeled as userType to specify which user is being referred to. We chose this approach because it allows us to store all the information about a user in a single database while still allowing for differentiation between the different users. Because each type of user has at most two unique attributes, the structure wouldn't be too complex. When a spot in the database has no value, (i.e. IT technician not having a location value), a null value is placed in the record for that user type.
Additionally, the same decision was then applied to our Postings Database. Because postings consisted of our system's notices, reports, and events, I applied the same reasoning because inheritance was also involved with these databases and using a identifier attribute simplifies down the number of databases while keeping the differentiation between these postings. We weren't too sure if the trails should even have its own database because there aren't too many trails in the park and databases are for handing large amounts of data. We implemented it in because HikePlan uses the trailIDs to differentiate all the hikes of each given time.

Figure 10: Mount Kilimanjaro's Summit
https://www.thecommonwanderer.com/blog/climbing-mount-kilimanjaro-things-to-know