“Kili Trekker System” Software Design Specification

Sonia Mossalaei, Michael Little, Hamza Dalloul

November 8, 2024

Table of Contents

1 System Description:

Brief overview of Trekker System:

The Kili Trekker System is a tracking and information management system developed to support the tourism business in the Kilimanjaro National Park, Tanzania. The system will help manage and track trekkers, provide vital information about weather conditions, park events, and emergency situations. The system aims to enhance safety, make the planning process simpler for both trekkers and guides, ensure a seamless experience for visitors, and make management of the park more efficient for park authorities.

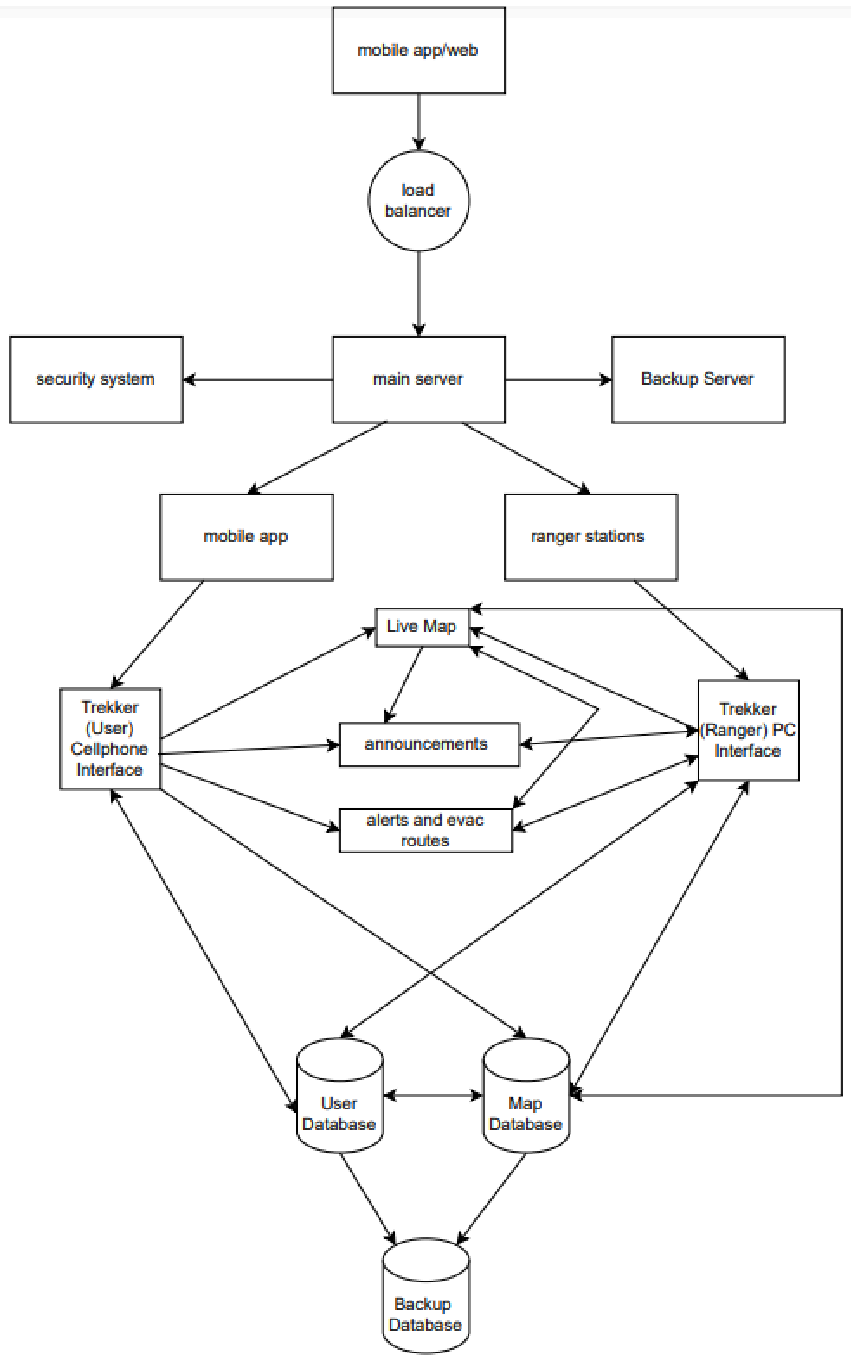
Mount Kilimanjaro is one of the most popular tourist destinations in Tanzania, attracting thousands of trekkers every year. Due to its vast trails, changing weather conditions, possible risks, and emergency situations, a centralized system is necessary to manage information and individuals in the park to ensure the safety of all visitors at all times. The Kili Trekker System will allow park rangers, trail guides, and other officials to access and update crucial information about the 12 different trails, events taking place, and emergencies. Additionally, it will provide a platform for trekkers to view current trail conditions and plan their trips accordingly. The trekkers are mandated to use the app while on trails and nearby areas to ensure safety and allow for them to receive real-time updates. This Design Specification Document outlines the architecture of the Kili Trekker System. The document is organized as follow:

1. System Description: brief overview of the Kili Trekker System
2. Software Architecture Overview: architectural diagram of all major components and UML Class Diagram
3. Development plan and timeline: description of the development plan and roles of each software engineer with timeline

This document will guide the development team by providing a clear design framework and ensuring that the system aligns with stakeholders needs while being technically sound.

2 Software Architecture Overview

2.1 Architectural Diagram



Description of components and their connections

The system will be available both as a mobile application and on the web. The trekkers will be mainly using the mobile app in order to plan their trip, stay up to date with announcements and events, and get live updates about the trails while at mount Kilimanjaro. The rangers, park staff and trail guides will be utilizing the web application in order to access the system (read/write/delete). Since there can be multiple users using the system simultaneously, there is a load balancer in order to distribute the network traffic across different services to improve the performance of the system. The main server represents the multiple servers that will be in use. There are backup servers available in case of any technical difficulties or more than the usual network traffic. Additionally, there will be a security system in place in order to avoid Kenyan rebels entering incorrect information into the system or disabling it entirely.

The main server connects to both the mobile app, used by trekkers, and the ranger station, which is the web application used by authorities, park staff, and rangers at the Tanzanian Kilimanjaro Park headquarters located at the main entrance.

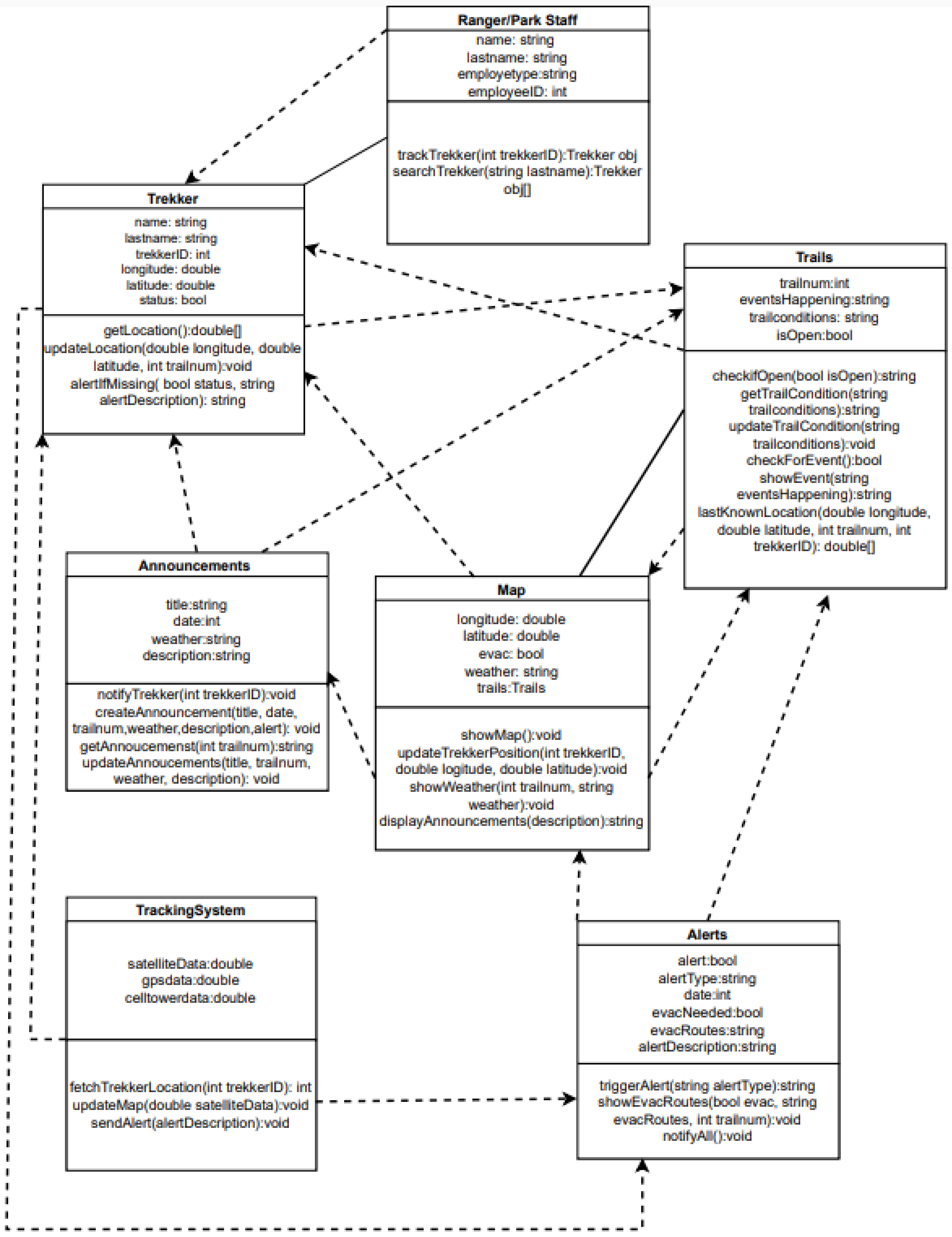
The mobile app will be connected to the Trekker(user) cell phone interface which is accessing the live map, announcements, and alerts and evac routes. When the trekker creates their account on the mobile app, their information such as name, last name, trakkerID, longitude, latitude, and status will be stored in a table in the User Database. The UI can also pull this information from the database in order to display it for the trekkers on their profile page. The trekkers locations (longitude and latitude) will also be stored in the Map Database in order to be accessed by the live map.

The ranger station has a monitor that accesses the web application. The trekker(ranger) PC interface also has access to the live map, announcements, and alerts and evacuation routes. Additionally, the rangers have the ability to make edits in the live map, announcements, and send out alerts and evacuation routes when necessary, hence why double sided arrows are used for rangers (including park staff, trail guides). Since rangers are also a user of the system their information will also be stored in the User Database and their location will be stored in the Map Database.

The Live Map includes information such as trail numbers, trail updates, live weather updates, and an actual visual of mount Kilimanjaro. The Live Map has access to the announcements and alerts and evacuation routes in order to present this information in real time to users on the trails and nearby areas of mount Kilimanjaro. The Live Map pulls data and sends data to the Map Database, information such as trekker longitude and latitude can be pulled from the Map Database when trekkers go missing and their last known location is needed for alerts (double sided arrow from alert to live map for this reason).

Additionally, there is a backup database in case of technical difficulties, system failures, etc. The backup database won't keep as detailed information as the individual databases, but it will backup important information from both databases at the end of each day.

2.2 UML Class Diagram


Ranger/Park Staff
name: string
lastname: string
employetype:string
employeeID: int
trackTrekker(int trekkerID):Trekker obj
searchTrekker(string lastname):Trekker obj[]
Trekker
name: string
lastname: string
trekkerID: int
longitude: double
latitude: double
status: bool
getLocation():double[]
updateLocation(double longitude, double latitude, int trailnum):void
alertIfMissing(bool status, string alertDescription): string
Trails
trailnum:int
eventsHappening:string
trailconditions: string
isOpen:bool
checkifOpen(bool isOpen):string
getTrailCondition(string trailconditions):string
updateTrailCondition(string trailconditions):void
checkForEvent():bool
showEvent(string eventsHappening):string
lastKnownLocation(double longitude, double latitude, int trailnum, int trekkerID): double[]
Announcements
title:string
date:int
weather:string
description:string
notifyTrekker(int trekkerID):void
createAnnouncement(title, date, trailnum,weather,description,alert): void
getAnnoucemenst(int trailnum):string
updateAnnoucements(title, trailnum, weather, description): void
Map
longitude: double
latitude: double
evac: bool
weather: string
trails:Trails
showMap():void
updateTrekkerPosition(int trekkerID, double logitude, double latitude):void
showWeather(int trailnum, string weather):void
displayAnnouncements(description):string
TrackingSystem
satelliteData:double
gpsdata:double
celltowerdata:double
fetchTrekkerLocation(int trekkerID): int
updateMap(double satelliteData):void
sendAlert(alertDescription):void
Alerts
alert:bool
alertType:string
date:int
evacNeeded:bool
evacRoutes:string
alertDescription:string
triggerAlert(string alertType):string
showEvacRoutes(bool evac, string evacRoutes, int trailnum):void
notifyAll():void

Description of Classes

Ranger/Park staff: The ranger/park staff class will keep track of the names, last names, employee type, and employee ID. The Ranger class will be able to keep track of trekkers so monitor visitors and ensure safety. This class uses the Trekker class to retrieve trekkerID for its operation and has Trekker for the list that is returned by the searchTrekker operation.

Trekker: The Trekker class keeps track of trekker profiles, name, last name, trekker ID, their location, and their status. The Trekker class can give the location of the trekkers and alert if they are missing. The trekker uses Trails and Alerts for its operations.

Map: The map class revolves around the live terrain and topography of the hiking area, and any events occurring within. This class has Trails, for trail numbers, and it uses Announcements and Trails for its operations.

Announcements: The announcements class has a main functionality of informing trekkers in the area about ongoing events. The announcement class creates and updates the announcement. This class uses Trekker and Trails for its operations.

Trails: The trails class keeps track of trail numbers and gives relevant information about the trails such as events happening, trail conditions such as weather, and whether the trail is open. This class uses Map and Trekkers for its operations.

Alerts: The Alert class sends emergency information to the trekkers in the area, as well as the necessary details thereof.Alerts can be triggered, evacuation routes can be shown and a notification can be sent to all users. This class uses Trails and Map for its operations.

Description of Attributes

Ranger/Park Staff Class:

- name: *string* consisting of first name of park employee
- lastName: *string* consisting of last name of park employee
- employeeType: *string* referring to employee's job title
- employeeID: *int* that contains unique employee identification number

Trekker Class:

- name: *string* consisting of first name of park hiker

- lastName: *string* consisting of last name of park hiker
- trekkerID: *int* that contains unique hiker identification number
- longitude: *double* measurement of hiker's location (east or west)
- latitude: *double* measurement of hiker's location (north or south)
- status: *bool* to mark hikers that are lost, missing, or need help/assistance

Map Class:

- longitude: *double* measurement for map locations (east or west)
- latitude: *double* measurement for map locations (north or south)
- evac: *bool* that signifies an evacuation for a trail#
- weather: *string* describing current weather conditions on each trail
- trails: *Trails* has the Trails class attributes and operations

Announcements Class:

- title: *string that* contains the title of the announcement
- date: *int* that contains the date of the announcement
- weather: *string* that contains important weather condition changes
- description: *string* that contains the announcement body/description
- alert: *boolean* that states whether the announcement is an alert(true) or not (false)

Trails Class:

- trailnum: *int* that contains the trail identification number
- eventsHappening: *string* contains current events going on at certain trails
- trailconditions: *string* that contains a trail's conditions
- isOpen: *bool* true signifies an open trail, false signifies closed trail

Alerts Class:

- alert: *bool* that signifies an active parkwide alert
- alertType: *string* that specifies the type of alert
- date: *int* that contains the date of the alert
- evacNeeded: *bool* that signifies if evacuation is needed
- evacRoutes: *string* that contains the recommended evacuation routes
- alertDescription: *string* that contains the description of the

TrackingSystem Class

- statelliteData: *double* containing satellite data
- gpsData: *double* containing gps data
- celltowerData: *double* containing cell tower data

Description of Operations

Ranger/Park Staff:

- **trackTrekker(int trekkerID):Trekker obj** This operation retrieves an object of type Trekker via the individual ID number on that trekker's app.
- **searchTrekker(string lastname):Trekker obj[]** This operation retrieves the list of all Trekkers obj with a certain registered last name

Trekker:

- **getLocation():double[]** This operation retrieves the location of any hiker via their coordinates and returns the location as a double
- **updateLocation(double longitude, new latitude, int trailnum):void** This operation takes the parameter of a trekker's location, as well as the trail number. This is so that the trekkers' most to date location is visible on the map.
- **alertIfMissing(bool status, string alertDescription)** This operation will be triggered if the status of the trekker is false, meaning that their location is unknown and hence missing. It will return a string with the trekkers information.

Announcements:

- **notifyTrekker(int trekkerID):void** This operation uses the attribute trekker ID from the Trekker class to only notify a specific trekker.
- **createAnnouncement(string title, int date, int trailnum, string weather, string description, bool alert):void** This operation creates an announcement with given parameters. The parameters include a title, the date the announcement is sent out, the trail in which the event is taking place, the live weather, a brief event description, and a flag for alert priority.
- **getAnnouncements(trailnum):string** This operation will display announcement details for any given trail. The trail number can be searched for annoucmnets. Additionally, if the location of the trekker shows that they are on that trail the announcement will appear on the map for the trekker.
- **updateAnnouncements(string title, int trailnum, string weather, string description):void** This operation updates details on a preexisting announcement, using the title, weather, trail number, and updated description.

Tracking System:

- **fetchTrekkerLocation(int trekkerID): double** This operation manually locates a specific trekker via their ID and displays their coordinates.
- **updateMap(double satelliteData):void** This operation uses satellite data to update an area of a map (i.e. a change in the topography).
- **sendAlert(string alertDescription):void** This operation uses the alertDescription attribute from Alerts to send an alert to all trekkers with a brief message regarding the situation.

Map:

- **showMap():void** This operation displays the live map on the device.
- **updateTrekkerPosition(int trekkerID, double longitude, double latitude):void** The trekkers location will be updated as they go through the trails and the surrounding mount kilimanjaro area. The longitude and latitude will be used to update their position to the most recent position.
- **showWeather(int trailnum, string weather):void** Displays the live weather conditions for a specific trail.
- **displayAnnouncements(string description):string** Passes the description attribute from the announcements class to display the details of an announcement.

Trails:

- **checkIfOpen(bool isOpen):String** Displays open when the attribute isOpen is true, and closed if isOpen is false.
- **getTrailCondition(string trailconditions):string** This operation returns the traversability conditions of a trail.
- **updateTrailConditions(string trailconditions):void** Updates the conditions of the trail, replacing any old information with new conditions.
- **checkForEvent()bool:** Displays a true or false value if an event is occuring on a trail.
- **showEvent(string eventsHappening):string** Provided an event is taking place, this displays details of live events in the park/on any trails.
- **lastKnownLocation(double longitude, double latitude, int trailnum, int trekkerID):double[]** This operation sets the last known location of a missing trekker, requiring their last known longitude and latitude coordinates, as well as the trail they were last seen on.

Alerts:

- **triggerAlert(string alertType): string** Sends out an alert from the ranger systems to all trekkers in the park, provided that the ranger passes the type of alert (i.e natural disaster)
- **showEvacRoutes(bool evac, string evacRoutes, int trailnum):void** Displays the nearest evacuation routes for all trekkers, with parameters of the value of the evacuation status, the nearest route, and the available trails
- **notifyAll():void** Used by the rangers to notify all trekkers about a specific emergency occurring

3 Development plan and Timeline

3.1 Partitioning of Tasks

There will be 3 software engineers working on implementing both the Mobile application and web development of the Kili Trekker system.

Sonia: Mobile Application Development
Week 1-2:

- Set up project repository
- Design the Mobile UI/UX for the trekker interface: onboarding, home page, map, announcements, and alerts
- Implement user authentication (create account/login) and integration with user database for trekkers

Week 3-4:

- Integrate real-time GPS tracking for trekker and integrate with Map Database
- Develop functionality for trekkers to view trail information, announcements, alerts, and real-time weather updates

Week 5-6:

- Add the ability to trekkers to update their status and last known location
- Implement a notification system for trekkers to receive alerts and announcements while not in the app

Week 7-8:

- Testing and debugging
- Ensure data syncing between the mobile application and web application

Michael: Web Application Development
Week 1-2:

- Design the web interface for rangers/trail guides/park officials: dashboard, map view, announcements, and alerts
- Implement user authentication for rangers, trail guides, and park officials

Week 3-4:

- Develop the live map functionality for tracking trekkers, trails, events, and weather
- Integrate with Map Database to fetch and display trekker locations, trail conditions, alerts, and announcements

Week 5-6:

- Add functionally for rangers to send alert and evacuation routes to trekkers via the web application
- Implement CRUD operations for announcements, alerts, and any other relevant updates

Week 7-8:

- Testing and debugging the web application
- Optimization for real-time map updates and notifications

Hamza: Backend and Database Development
Week 1-2:

- Set up the user and map database
- Define API endpoints for interaction between the mobile app, web system, and the databases

Week 3-4:

- Implement the live map tracking system: API integration for fetching and updating trekker locations from GPS data

Week 5-6:

- Implement alert and evacuation routes triggers, integrating with the Map Database and ranger inputs
- Create satellite data integration for map updates and topographical changes

Week 7-8:

- Set up security system to prevent unauthorized access and protect data from external threats
- Testing backend performance and security and proper functionality
- Ensure seamless data flow between system

3.2 Timeline

Weeks 1-2: Set up and begin UI, database, and backend development
Weeks 3-4: GPS tracking, map, announcement system, and basic app/web functionally will be implemented
Weeks 5-6: Alerts, announcements, notifications, and full system integration
Weeks 7-8: Final testing, debugging, and improvements

This timeline will allow each of the engineers to focus on their tasks while ensuring that the project progresses smoothly, The timeline accounts for the need to integrate components developed by other engineers and includes time for debugging, testing, and optimization.

4 Test Plan

4.1 Software Design Specification

We believe that our previous design for the architectural diagram completely covers the design specification of The Killi Trekker system. We did make a few modifications to our UML class diagram. Among those changes are:

- **The getLocation():void** function was changed to **getLocation():double[]**. The function output was changed from a double to a double[]. This would make the function able to display coordinates, longitude at index 0 and latitude at index .
- **lastKnownLocation(double longitude, double latitude, int trailnum, int trekkerID):double[]**. The function output was changed from a double to a double[]. This would make the function able to display coordinates longitude at index 0 and latitude at index .
- **createAnnouncement(string title, int date, int trailnum, string weather, string description, bool alert):void.** We added another parameter to signify an announcement as an alert. This assigns priority to the announcement, ensuring that all users see it displayed as an alert and don’t have to search in announcements for it.

4.2 Verification Test Plan

Unit testing:

1) This tests the getLocation() operation. The operation returns an array of doubles of size 2. At index 0 longitude is returned and at index 1 latitude is returned. The test makes sure that index 0 is the same as the longitude stored in L and that index 1 is the same as the latitude stored in L.

Trekker getLocation():double[]

```
Location L
L.name = John
L.lastname = Doe
L.trekkerID = A12345
L.longitude  = 123.45
L.latitude = 678.90
L.Status = true
if(L.getLocation().double[0] == L)
Return pass
Else
Return fail
if(L.getLocation().double[1] == L)
Return pass
Else
Return fail
```

2) This tests the isOpen() operation. If trail is open it will display a string stating the number of the trail and whether it's open or closed based on the isOpen attribute. For this test isOpen is true, trail is open, so trail 9 is open and should be displayed to pass the test.

Trail isOpen(bool isOpen):string

```
Trail t
t.trail num = 9
t.eventsHappening = puppet show
t.trailconditions = good
isOpen = true
if(t.isOpen(isOpen) displays “trail 9 is open”)
```

```
Return pass
Else
Return fail
```

Integration Testing:

1) This integration test verifies the functionality of the showWeather(trailnum, weather):void. This operation uses Trails in order to get the trail number and get weather conditions from the Map. If the weather is displayed for users then it passes the test, if it is not displayed for the users then the test fails.

showWeather(trailnum, weather):void (Trails class + Map class)

```
Trail t
t.trailnum= 8
t.eventsHappening = no events
t.trailconditions =  slippery
t.isOpen = true
showWeather sw
weather = raining
if(sw is displayed)
return pass
Else
Return fail
```

2) This integration test ensures that the lastKnownLocation operation from the trails class functions as intended with an ID integer from the Trekker class.

lastKnownLocation(double longitude, double latitude, int trailnum, int trekkerID):double[] (Trails class + Trekker class)

```
Trail t
t.trailnum= 8
t.eventsHappening = no events
t.trailconditions =  slippery
t.isOpen = true
Trekker tk
tk.trekkerID = A12345
Double[] trekkerLocation = [];
if(t.lastKnownLocation(longitude, latitude, trailnum, trekkerID).double[0] ==t)
Return pass
Else
```

```
Return fail
if(t.lastKnownLocation(longitude, latitude, trailnum, trekkerID).double[1] == L)
Return pass
Else
Return fail
```

System Testing:

1) User enters the park area and checks for events
   User downloads the application and makes an account, wherein they input their name. Upon their entrance into the park area, the app will then begin to display their live location on an interactive map. From there, they check the weather on trail 12, any announcements, and then any events of interest on trail 12. If a live event is displayed, system success.

```
currentWeather = “Raining”
currentAnnouncements = “New activities on Trail 12!”
areEventsHappening = true
currentEvents = “Cultural exploration on Trail 12 at 2:30!”
showMap():void
showWeather(12, string weather):void
Weather = “Raining”
getAnnouncements(12):string
Announcements = “New activities on Trail 12!”
checkForEvent():bool
checkForEvent = true
showEvent(string eventsHappening):string
eventsHappening = “Cultural exploration on Trail 12 at 2:30!”
if (weather == currentWeather AND currentAnnouncements == announcements AND
areEventsHappening == checkForEvent AND currentEvents == eventsHappening):
return pass
else return fail
```

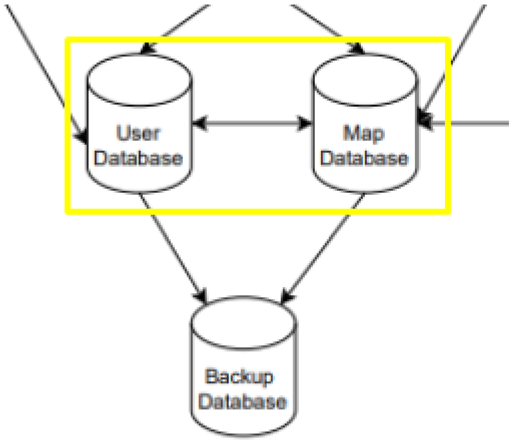
2) Admin responding to missing family report test case. Admin/Ranger receives a report for a missing family, giving their shared last name. Ranger then searches for their last known location using an operation, and identifies the trail they were last on and any important trail information. Using this data, the Ranger then creates an announcement with all important information, and triggers a high priority alert and

notifies all users of the missing family. Test will verify that information is displayed correctly and accurately on the user's ends.

```
lastName = "Vines";
Trekker[] missingPersonsInFamily = searchTrekker("Vines");
createAnnouncement("Missing Family!", 1024, 12, "flooded", "Vines family was last seen on
Trail 12. Note that Trail 12 has flooded conditions. Please send any valuable information so
we can locate them and give aid", 1)
triggerAlert("Missing Family")
notifyAll()
If (displays alert correctly on User's side as intended)
return pass;
else
return fail;
```

5 Software Design 2.0

5.1 Software Architecture Diagram



The software architect diagram was updated so that the User database and the Map database have access to one another. By connecting the two databases the tables can be related to one another and there can be foreign key relations between the tables. For example, the User Location table in Map database uses the UserID from the User table in the User database, hence why the databases need to be connected to one another as well.

5.2 Data Management Strategy

Data Management Strategy

We have decided to use SQL to manage our data due to its structured, relational data model, and its support of foreign key relationships. SQLs ACID(atomicity, consistently, isolation. durability) compliance helps ensure secure and reliable data management for user profiles, trail information, and location data. This aligns well with our requirements for managing sensitive and relational data. The tables will require relations to other tables and SQL allows for foreign key relations hence why we chose SQL for our database.

How many databases we chose and why:

We chose to have three databases, a **user database** that stores user and employee profiles, a **map database** that manages location-based data, as well as a **backup database** that will keep important information and it is only updated at the end of the day. Both the databases use one backup database. Information such as user profiles, employee profiles will be kept in this database as well in case of technical difficulties, possible breaches and accidental deletion of information in the user database. We do not want users to lose access to their accounts due to technical difficulties. The last known location of users will also be stored rather than their real-time location that is stored in the Map database. The Map database will store information such as trail numbers, whether they are open or closed, and if there are events happening on the trail in a trails table. There will also be a table for the events happening on the trails.

Logically Splitting the Data

To prioritize efficiency and proper organization of the data, we created database tables specifically pertaining to the most important information being stored in our databases. After careful consideration, we deemed the most important aspects to be: user information (their names and ID's), employee information, live map information, and trail information. Using foreign key relations to perform inter-database access allows seamless cooperation between the tables.

USER DATABASE TABLES:

User Information Table:

UserID	First Name	Last Name	trekkerID
1313248	Hamza	Dalloul	846#
1313249	Sonia	Mossalaei	847#

Employee Table:

Employee Key	First Name	Last Name	Employee Type	Employee ID
1550	Miranda	Parker	Ranger	13252#
1552	Eli	Mishriki	Ranger	14528#

MAP DATABASE TABLES:

User Location Table:

UserID	Longitude	Latitude
1313249	32.775723	-117.071892
1313248	32.801420	-117.156540

Trails Table:

Trail Number (1-12)	Open?	Weather	Events Happening?
3	False	Stormy	False
12	True	Cloudy	True

Events Table:

Event Title	Type	Date	Time	Trail Number (1-12)
Memorial Day Sunrise Hike	Holiday Celebration	2025-05-26	05:30	7
Nature Photography Workshop	Educational Event	2025-01-11	15:00	3
National Trails Day Hike	National Park Event	2025-06-07	10:00	1

User Database Tables

- User table- stores genera user data like name, last name, and TrekkerID
- Employee table- stores employee records with data like name, employee type (ranger/park staff), and employeeID.

Map Database Tables:

- User location- stores real-time latitude and longitude based on UserID(foreign key relation to the User table)
- Trails table- contains trail number, trail status, weather conditions, and events happening
- Events table- stores the list of upcoming events, including event type, date, time, and trail number

Backup Database Tables:

- The backup database will also contain the user table, employee table, user location table, and the trails table.

- It will not be updated throughout the day and have real-time information.Updates will take place in the User and Map database multiple times throughout the day.

Possible Alternatives

1. **Using noSQL**

An alternative in technology would be using NoSQL for our database rather than using SQL. A NoSQL database would handle the dynamic and unstructured nature of real-time user location data and updates more flexibly. The data could be easily scaled horizontally across multiple servers, which could lead to improvement in the system's ability to handle large amounts of live/real-time data.

Tradeoffs:
Pros: NoSQL databases are optimized for unstructured data, such as location updates, which are used in this system. They can be quicker to write data and adapt easily if data models evolve.

Cons: NoSQL lacks the ability for relationships between tables that SQL offers. This complicates managing links between users, locations, trails, and events. It also doesn’t provide the same level of transactional support needed for sensitive information like user profiles

2. **Combining Databases**

An alternative to our data organization would be to combine the User and Map databases into one database. The data would be consolidated within one main database, while the backup database could still exist separately to store critical information.

Tradeoffs:
Pros: By combining the two databases this would simplify data management and eliminate the need for cross-database connections. Queries to retrieve user profile, location data, or event information are more straightforward as there is no need to perform cross-database joins or manage foreign key relationships between different databases. It is also easier to maintain. Any updates or migrations can be applied to one database, minimizing the risk of synchronization issues.

Cons: A combined database could lead to performance issues, especially when handling high read and write demands such as real-time location tracking and user profile management. As the data volume increases, more complex queries could experience delays, particularly if both user data and real-time location data are involved. It could also limit the ability to scale each data type independently. For example, if the real-time location data grows significantly, it could affect the performance for user-related queries as well. A combined database could also lead to longer recovery times in case of failure due to the larger data volume.