Kili Trekker System

By Ian Carpizo, Kyle Krueger, Juan Morales, and Fahri Yildiz

<u>CONTENTS</u>

1.0 User Manual
This user manual will attempt to provide clarification to common use cases that arise from our system. If there are any questions not answered by this manual contact our toll free support number at 1-800-901-1102.

1.1 User Account

1.1.1 Account Creation
In order to create an account, begin by navigating our home page to the top right corner to a button saying “Log In / Sign Up”, and this will take you to the account creation page.
The page will ask some questions about your name and email address, as well as creating a password.

1.1.2 Administration Account Creation
For creating an administration account, follow the steps listed in section 9.1.1, and once an account has been created, contact the park ranger’s station in order to gain administrative access.

1.2 User Homepage

To access the user homepage, you must first sign in by clicking on the “Login / Sign Up” button at the top right corner of the homepage. Once logged in, click your username at the top right corner in order to access the user homepage

1.2.1 Tour History
To access tour history, follow step 1.2 to access the user homepage, then click “download tour history” in order to download your previous tours and purchases as a PDF.

1.2.2 Password / Email Change
To change your password or email, follow step 1.2 to access the user homepage, then click “request password change” or “request email change”, and follow the instructions emailed to your account. Note that all official emails will come from admin@kilitrekker.com. We will not send promotional emails from this address, and please double check the email address before clicking on the email.

1.2.3 Accessing Administrative Home
In order to access the administrative home, you must have an administrator account (see 1.1.2) and if you are logged in as an administrator, then the button for Administrator Home will appear under password change. For more information on Administrative features, please contact the park ranger station.

1.3 Local Information

Our local information page will be featured on the right hand side of our homepage. It should display weather information, and whether the trails are currently opened or closed. Clicking on the “Local information” button will redirect you to the local information page.

1.3.1 Emergency Situations
If for some reason the park is unsafe or unavailable to visit, the Emergency Situations page will provide more information. Note that it will be displayed on the homepage whether the park is closed or open.

1.4 Trail Information

Clicking on the Trail information button on the home page will direct you to a list of the trails and the information about them, including distance, hours of availability, and a link to a map of the trail.

1.4.1 Booking a tour based on a certain trail
One of the ways you can book a tour is from the trail information page. When viewing the list of trails, there will be a link to take you to the available tours based on that trail. This link will be on the right side of the trail information page, next to the respective trail, and will redirect you to the tour page for that trail.

1.5 Tour Page

The tour page will show available tours to be booked. Tours will be booked through the tour company’s website, not through kilitrekker.com. The page will sort the tour companies by rating, and there are filters at the top of the list that can be applied to sort the tour by duration, trailhead, time of day, and distance.

1.5.1 Booking a tour
Booking a tour from the tour page is simple. Simply find the tour that you would like to book on the tour page, and to the right of the company, there will be a hyperlink that says “book now”. This will redirect you to the respective company’s tour booking page, and once you get a confirmation that your tour has been booked, your tour history page on your profile will be updated to reflect your purchase.

1.5.2 Cancelling a tour
To cancel a tour, navigate to the tour history page (follow steps on 1.2.1) and click on “request cancellation”. This will redirect you to the company page, and will be reflected on the tour history page once the cancellation is confirmed.

2.0 System Description

The firm has been requested to create a system to provide information to the visitors coming to Mt. Kilimanjaro National Park, titled the “KiliTrekker” system. The system will display information about trails in the park, local information, local Kenyan rebel activity, and tour information . Our system will be used by visitors of the park as users, and administrator access will be given to park rangers and park staff. The administrators will update the details of the area, including local events and celebrations, Kenyan rebel activity, tour company information and tour info, and weather will be provided by an external service. It will be available to be accessed through any browser, including mobile support, and admins may update information using the park ranger’s PC which will be hosting the system. This document’s primary focus is to develop a high-level overview of the proposed system outlining its components, users, and attributes. In addition, this document will provide an architectural overview of the proposed system detailing the interactions between the various components of our system.

2.1 Validation

Our system meets the requirements put forward by the firm. Our users are able to securely navigate a website that is able to provide information on the area of the trails and include information about weather conditions and rebel activity of the area. Users are able to schedule tours with local tour guides, which are posted by various tour companies. Administrators can update local information such as rebel activity, and tour companies can update their available tours and contact information. Our system is secure, and we talk about security in section 9 of this document. Our system requires two databases to be set up at the home ranger station at the park, and these data bases contain basic user data and tour company information. Our website is accessible through any device with a web browser, which includes personal computers and mobile devices. Requirements can be found in section 2.0 above.

3.0 Software Architecture Overview

This section outlines the components of our system in a UML diagram. The diagram shows the classes/entities, their attributes and methods used in completing the desired tasks. The system contains classes and variables for all of our interactable sections in our system, including the home page, local information page, tour information page, administrator home page, and trail information page. The users will enter the website through the homepage, where they can navigate to their desired web page depending on their task. Admins will enter the website through their own homepage, and can update information on the other pages through it. The trails contain details of treks and interesting areas. The homepage contains information on the safety of the trail, weather, and buttons to go to the tour, trail, and local information pages.

In the next few pages of the document, we will be analyzing our architecture diagram and discussing the purpose of each class. These descriptions were written by the authors of the diagram, and have specified the functionality of each variable and function within.

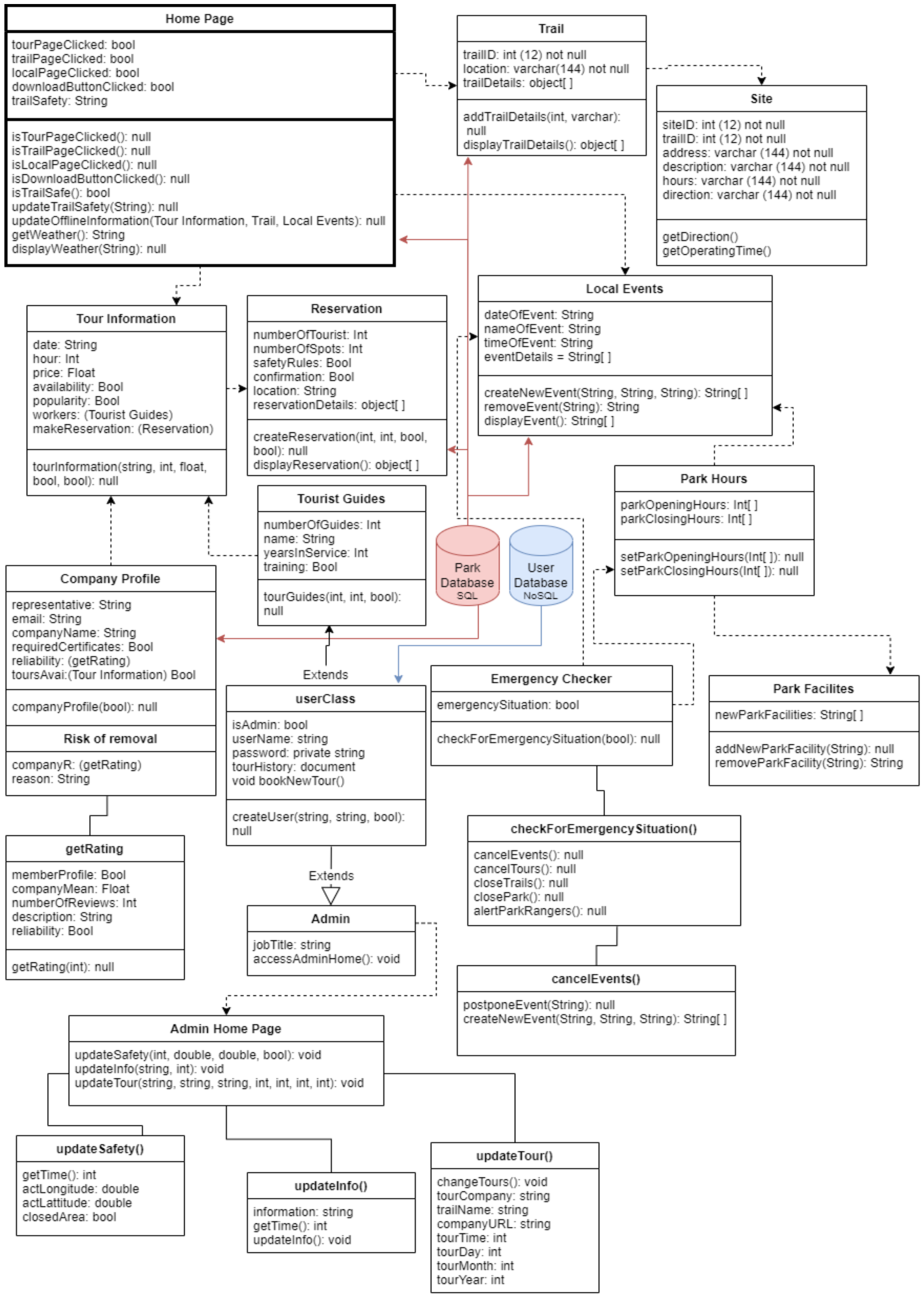

Figure 3.0.1 Architecture Diagram

3.1 Homepage

Here in the figure above, the attributes and operations for the home page are displayed. This class is meant to be a link between three of the classes: the “Tour Information” class, the “Trail” class, and the “Local Events” class. The attributes will include a checker to see if either the tour page, trail page, or local information page have been clicked, a checker to see if the download button has been clicked, and what the current status of the trail’s safety is. The first four attributes (“tourPageClicked”, “trailPageClicked”, “localPageClicked”, and “downloadButtonClicked”) function the same way. These variables will be boolean and be set initially to false, and will only be set to true if they’re respective link or button has been pressed. The last attribute is the variable “trailSafety”, which will be a string in order to contain the status of the safety of the trail.

Moving on to the operations, the first four operations (“isTourPageClicked()”, “isTrailPageClicked()”, “isLocalPageClicked()”, and “isDownloadButtonClicked()”) relate to the first four attributes. These four operations will constantly keep checking their respective attribute, and once that attribute is set to true, then it will either redirect the user to the page they selected, or begin downloading the offline version of the home page. The next operation, “isTrailSafe()”, will check to see what the current value of “trailSafety” is, and if it is a string representing that the trail is safe, then it will return true, but if the string is representing that the trail isn’t safe, then it will return false and alert the park rangers. The function “updateTrailSafety()” takes a string as a parameter and uses that string in order to update the value of “trailSafety” to reflect the new string that was passed in.

The method “updateOfflineInformation()” will take three parameters, each of the type of their respective classes: “Tour Page” type, “Trail” type, and “Local Events” type. These three parameters will be used to update the information attached to each of the three classes. For example, this method could be used to update a new event that is coming up by calling the “createNewEvent()” function from the “Local Events” class. Similarly to the “Local Events” class, the function “getWeather()” will also call the API being used in order to take the current weather from Google, and will return a string that will convey how the current weather is, such as “Sunny”. Using this string, the last method, “displayWeather()”, will take that string as a parameter in order to display the current weather on the home page.

3.2 User and Admin functions

The class titled “userClass” collects information about the user solely for the user’s benefit, as we do not directly deal with payment information. We should collect information about their tour history and what their booked tours, and to do this the user should set a login with a username and password. If the user is an administrator, the variable isAdmin will be true, allowing access to the Admin Home page, which has the functions of updating the various pages on the website.

The admin home page allows us to update safety information. Administrators should put in the longitude and latitude of recent rebel activity, which could be either manually entered or selected from a map. Admins should also have the activity of shutting the park down virtually by updating the closedArea variable to true.

The updateInfo function has the task of updating the park information page, which is saved on a string, and whenever the information is updated, we should include a timestamp of the time it was last updated, and we should get the time from the local computer time.

Finally, from the admin home we should be able to use the updateTour function in order to update information on our tour page. Tour information will be collected from tour companies through either electronic or interpersonal means. Information input into the system about upcoming tours should include the company providing the tour, the trail that it is on, and the time of the tour and we will provide a link to their website if one is available.

3.3 Local Events Page

As you can see from the figure above, there will be a set of classes containing all of the information pertaining to the local area of the Kilimanjaro National Park. The information being stored in these classes shall include the dates, names, and times of the local events, the park's opening hours, the park's closing hours, an emergency situation checker, and the new park facilities being constructed. Starting with the "Local Events" class, there will be three attributes, each stored as strings: the date of the event ("dateOfEvent"), the name of the event ("nameOfEvent"), and the time of the event ("timeOfEvent"). If a new event were to come up, then the function "createNewEvent()" would be called, which takes three strings as parameters. The three strings that would be entered would be the three attributes of the "Local Events" class. Once the strings have been passed in, the function will create an array containing the date, name, and time of the particular event. The other method, "removeEvent()", will take a string as a parameter, search for said string in the array created by "createNewEvent()", remove that string from the array, and then return the string once it's complete.

Next, the "Park Hours" class contains the attributes of the park's opening and closing hours, which will both be stored as a list of integers, with their respective variables "parkOpeningHours" and "parkClosingHours". The lists will be ordered in such a way that the first element of the list will be the time for Monday, and the last element in the list will be the time for Sunday. The reason the opening and closing times are two separate lists is so that if either one needs to be changed, then the respective list can be changed, leaving the other unaltered. The two operations within this class are "setParkOpeningHours()" and "setParkClosingHours()". The two functions will both take a list of integers as parameters. Once both parameters have been specified, the functions will replace their respective attributes ("parkOpeningHours" and "parkClosingHours") with the new list specified.

The class "Emergency Checker" is meant to constantly keep checking whether or not there is an emergency in the park. The attribute inside this class is the "emergencySituation" variable, which will be a boolean in order to portray whether or not an emergency situation is occurring in the local area. If there is no emergency happening, then the variable will remain "false", but in the case of an emergency, the variable will be switched to "true". The next function, “checkForEmergencySituation()”, will be used to constantly keep checking the “emergencySituation” variable. Normally, this function will return “false” since the “emergencySituation” variable will typically be false, but in the case that the variable is set to true, then the function will return true and begin alerting the nearby rangers and tourists that an emergency situation has just occurred. In addition, it will run the functions “cancelEvents()” and “closePark()” in order to cancel any events happening that day and shut down the park. In particular, once “cancelEvents()” executes, it will call the operation “postposeEvent()”, which takes a string as a parameter, in order to take that event out of the list of events. In addition, “createNewEvent()” will execute as well in order to create a new date and time for the event that was recently canceled.

Last, the “Park Facilities” class will contain the “newParkFacilites” variable, which will contain a list of strings, holding the name of each of the new park facilities currently being constructed. The “addNewParkFacility()” function will take a string as a parameter, and will simply add the string to the “newParkFacilites” list of strings. The “removeParkFacility()” function will do the exact opposite: it will take a string as a parameter and search through the “newParkFacilites” variable, and once it has found the string, it will remove it from the list and return the string.

3.4 Tour Page

In this section the description of the Tour class diagram will be described and explained. The start point is the description of the company profile where every detail of each company available for the system will be displayed. Some important variables are the name of the company's representative and the name of the company. An email and required certificates will also be necessary. Then, two functions will be called; getRating and a special separate function for the tour information. getRating will be a crucial part of this system because users will have the opportunity to pick the guide company that best fits their needs. This function will have some variables, such as member profile, company rating mean, number of reviews, description of the reviews, and company reliability. On the other hand, there is a dedicated function for the tour information where basic information, such as date, hour, price, and availability will be stored. In addition to this, two other helper methods will be implemented as part of the class: reservations and tourist guides. Reservations will contain private information from the user. The number of tourists and number of spots will be stored as integers and some special statements, such as safety check in and reservation confirmation, will be stored as booleans. To conclude, two special attributes will be added to the system. One of them is the “risk of removal” attribute, where the program will determine if the company will withdraw based on negative ratings, and the “cancelR” attribute, where the user will have the opportunity to cancel the reservation.

3.5 Trails Page

The trails class contains the descriptions of the 12 trails. The site class describes details about each location. These details include address, directions and operational hours. These attributes are defined by their respective variables depending on the required allocations for storage in the database. The description offers insight into the events while the status shows whether there is danger or not. The administrators can view events and update events. Users can view events only.

3.6 Databases
Our system's data will be contained on two databases. The first database will be the Park Database, indicated in red on *Figure 3.0.1*. The park database will contain the company database, local database, trail database, whether database, and availability database.
Our second database will be the user database, indicated in blue on *Figure 3.0.1*. More information about the database can be found in the diagram in section 6 of this document.

4.0 Development Plan and Timeline
The proposed project will be completed in the next 42 days. During this duration, each team member will be responsible for tackling different tasks in order to complete the Kili Trekker System. In the table below, every task will be laid out clearly, including the amount of time it will take to complete, as well as the person(s) who will be in charge of completing the task.

Kili Trekker System Plan			
Milestone	**Activity**	**Duration (Days)**	**Responsibility**
Initiation		**10**	
	Requirements gathering	7	Kyle Krueger and Fahri Yildiz
	Specification detailing	1	Ian Carpizo
	User requirements documentation	2	Juan Morales
Planning		**4**	
	Scheduling	2	Ian Carpizo
	Resources allocation	1	Fahri Yildiz
	Template development	1	Juan Morales
Execution		**20**	
	Coding Homepage	4	Ian Carpzio
	Coding User and Admins	2	Kyle Krueger
	Coding Info Page	3	Ian Carpzio
	Coding Trail Page	3	Fahri Yildiz
	Coding Tour Page	3	Juan Morales
	Prototyping	3	Fahri Yildiz
	Initial Release	2	Ian Carpizo
Monitoring		**3**	
	Testing	2	Juan Morales
	Feedback collection and implementation	1	Fahri Yildiz
Closure		**5**	
	Final Testing	1	Kyle Krueger
	Deployment & Documentation	2	Juan Morales
	User training	2	Ian Carpizo
	Total	**42**	

5.0 Verification Test Plan

In this section, we will portray several different black-box tests we performed on our Kili Trekker System. The following tests utilize changes that were made to the UML Class Diagram that are explained in Section 6.1. In total, there are three unit tests, two integration tests, and two system tests outlined in the following subsections. Each test displays inputs with expected outputs, showing the flow intended for our functions described in our UML Class Diagram.

5.1 Unit Tests

5.1.1 Unit Test #1

setParkOpeningHours(Int[ ]): null

- This function will take in a list of integers that will be ordered where index 0 corresponds with Monday and index 6 corresponds with Sunday
- This function won't return anything, but rather will update the variable "parkOpeningHours" within the "Park Hours" class, so that is what will be tested
- The hours will be represented as they would be normally, except without the ":", so "9:00" would just be "900"
- Hours between 12:00pm and 12:00am will be represented with military time, so 1:00pm would be 13:00, and 12:00am would be 0:00
- If less than 7 elements are present within the array, the remaining elements will be set to 0
- If more than 7 elements are present within the array, only the first 7 will be accepted, deleting the remaining elements

Test 1 (7 elements):
<[900, 900, 900, 900, 1000, 1200, 1300], park.parkOpeningHours>
park.parkOpeningHours = [900, 900, 900, 900, 1000, 1200, 1300]

Test 2 (5 elements):
<[900, 900, 900, 900, 1000], park.parkOpeningHours>
park.parkOpeningHours = [900, 900, 900, 900, 1000, 0, 0]

Test 3 (10 elements);
<[900, 900, 900, 900, 1000, 1200, 1300, 1400, 1500, 1600], park.parkOpeningHours>
park.parkOpeningHours = [900, 900, 900, 900, 1000, 1200, 1300]

Test 4 (0 elements):
<[ ], park.parkOpeningHours>
park.parkOpeningHours = [0, 0, 0, 0, 0, 0, 0]

5.1.2 Unit Test #2

setParkClosingHours(Int[ ]): null

- This function will operate the same as "setParkOpeningHours(int[ ]), except instead of updating the "parkOpeningHours" variable, it will update the "parkClosingHours" variable

Test 1 (7 elements):
<[2100, 2100, 2100, 2100, 2030, 1900, 1900], park.parkClosingHours>
park.parkClosingHours = [2100, 2100, 2100, 2100, 2100, 2030, 1900, 1900]

Test 2 (3 elements):
<[2100, 2100, 2100], park.parkClosingHours>
park.parkClosingHours = [2100, 2100, 2100, 0, 0, 0, 0, 0]

Test 3 (12 elements):
<[2100, 2100, 2100, 2100, 2030, 1900, 1900, 400, 800, 900, 600, 500], park.parkClosingHours>
park.parkClosingHours = [2100, 2100, 2100, 2100, 2100, 2030, 1900, 1900]

Test 4 (0 elements):
<[ ], park.parkClosingHours>
park.parkClosingHours = [0, 0, 0, 0, 0, 0, 0]

5.1.3 Unit Test #3

addTrailDetails(int, varchar): null

- This function takes in an integer and a varchar as parameters and updates the following variables: “trailID” and “location”

<10, “Kilimanjaro National Park”, trail.trailDetails>
trail.trailDetails = [10, “Kilimanjaro National Park”]

5.2 Integration Tests

5.2.1 Integration Test #1

Our first integration test will represent the user browsing for a tour and creating a reservation. We will start by navigating to the tour information page:

tourInformation(string date, int hour, float price, bool availability, bool popularity)

This function will take several arguments:

- A date that will be a string with an hour time and a price that will be int and float respectively
- Then, two conditions will be tested (bool argument) to check if the tour is available or not and how popular it is.
- Finally, once we have chosen to browse tours, three functions will be called: tourGuides(), createReservation() and companyProfile().

  <date, hour, price, availability, popularity, results>
  <”8-18-22”, 900, 129.99, true, true, display option to create reservation>
  <”3-23-22”, 1000, 89.99, true, false, display option to create reservation>
  <”11-08-22”, 1215, -1, true, true, Error: null price>

tourGuides(int numberOfGuides, int yearsInService, bool training)

- This class will receive the number of guides available with the years in service as an ints and then provide the name of the tourist guide as a string.
- Finally, a condition must be complete to transform the false statement to true. This will represent the training that a tourist guide must complete for verification of abilities.

  <numberOfGuides, yearsInService, training, output>
  <5, 10, true, Proceed>
  <2, 3, false, Reject>

companyProfile(bool requiredCertificates)

- This function will display the name of one representative with an email and the company name.
- Then, a condition will be tested for the requiredCertificates: if(requiredCertificates == false)
- This method contains the function getRating() and is referenced in the "Tour Information" class when navigating to the tour information page.

  <requiredCertificates, test result>
  <true, display company>
  <false, don't display company>

float geRating(int numberOfReviews)

- In this function, an operation will be implemented to get the rating of the company.
- First, the function will receive an int that will represent the number of reviews. This number will divide the rate that customers will input (a number between 1 and 5).
- Finally, customers will be able to input a written description.

  <numberOfReviews, add rating, expected output>
  <1, 3, 9>
  <-1,-1,NULL>

createReservation(int numberOfTourists, int numberOfSpots, bool safetyRules, bool confirmation)

- This function will display the number of tourists available from the function tourGuides().
- Then the number of spots available will be received as an int.
- Finally, two conditions will be tested: if(safetyRules == true) and if(confirmation == true). Then, if both conditions are true, the location will be displayed.

  <numberOfTourists, numberOfSpots, safetyRules, confirmation, expected output>
  <20, 30, true, true, 20 tourists booked>
  <15, 2, true, true, unable to book>

<20, 30, false, true, please complete safety program>

5.2.2 Integration Test #2

Creating a new administrator and adding a tour.

In order to create a new user, we must call the create user function from the user class and pass in the parameter of creating an administrator. We should pass in the userName and password as strings, and a true statement for the isAdmin variable.

Function 1
userClass.createUser(userName, password, isAdmin)

Once an administrator is created, our user should navigate to the admin home page and add a tour. Adding a tour requires the name of the tour company, the name of the trail, and the URL of the tour company as strings, and the tour date/time as ints which are given to the program for the day, month, year,and the time will be represented as they would in a normal digital clock, but without the semicolon (ie 9:30 would be 930).

Function 2
adminHome.updateTour(tourCompany, trailName, companyURL, tourTime, tourDay, tourMonth, tourYear)

These two functions will undergo the following test cases to ensure their reliability and to help us locate any faults and failures.

Tests for Function 1:
<userName, password, isAdmin, Expected Output>

<”Brandon Smith”, “AppleBottomJeanz12%”, true, New admin created>
<”Jim Henderson”, “BootzW!thTh3Fur”, false, New user created>
<NULL, “TheWh0!3Club”, true, Error: Username Cannot be Blank>
<”Wes Lukingather”, NULL, true, Error: Password Cannot be Blank>
<NULL, “$h3H1tTh3Fl00r”, false, Error: Username Cannot be Blank>
<”Ness Thinyano”, NULL, false, Error: Password Cannot be Blank>

Function 1 should create a new user regardless of the condition “isAdmin”, however it should create an administrator account if it is true. It should fail and give an error if the username or password does not meet the requirements.

Tests for Function 2:
<tourCompany, trailName, companyURL, time, day, month, year, Expected Output>

<”KiliTours”, “south trail head”, “www.kilitours.com/newtour”, 700, 12, 8, 2022, Tour updated>

<”Trailblazers INC”, “north trail head”,” trailblazersinc”, 800, 12, 3, 8, 2022, Error: Invalid URL>
<”Walkers”, “(invalid trail)”,” www.walkers.com/tours”, 1730, 23, 5, 2022, Error: Invalid Trail>
<”BeTrail”, “south trail head”, “www.betrail.com/booking”, 2500, 10, 12, 2021, Error: Invalid Time>
<”HighKings”, “turtle loop”, “www.highkingtrailtours.com/”, 1200, 12, 9, 1900, Error: Invalid Year>
<NULL, “turtle loop”, “www.kilitours.com/newtour”, 900, 10, 9, 2022, Error: Invalid company>
<”SafeTrails”, “north trail head”, “www.safetrails.org/tours”, 1300, 32, 9, 2022, Error: Invalid Day>
<”TrailBlazin”, “Ironman Trail”, “www.justblazit.com/tours”, 500, 28, -1, 2022, Error: Invalid Month>

Function 2 should only update the tour if all the information is valid. It doesn’t necessarily need to check if the information is correct, but rather that the information provided is in the correct format, and should indicate where the error occurred.

5.3 System Tests

5.3.1 System Test #1

This system test will demonstrate what happens when an emergency occurs in the park and everything is shut down. First, a user will be created that is an admin so that they can make changes to the data within the system. Next, updateTour() will be called in order to update the tour information. Then, reservations, events, and tour details will be added so that the system has data stored within it. Lastly, an emergency will occur, causing the “emergencySituation” variable to become true, calling the functions “cancelEvents()”, “cancelTours()”, “closeTrails()”, and “closePark()”.

createUser(string, string, bool): null

- This function will create a user for the software system.
- Three arguments will be received: a string for the username, a string for the password, and a bool that will represent the user type(admin or non-admin)
- Example 1: <true, Gerard Garcia, A123456, New admin user created.>
- Example 2: <false, Margaret Smith, B123456, New user created.>

updateTour(tourCompany, trialName, companyURL, tourTime, tourDay, tourMonth, tourYear): null

- This function can be accessed only by admin users.
- Several arguments will be received, including the name of the company, trial name, time, day, month and year.
- When, tourCompany is received the companyClass will be called in order to verify the company. Then, company information will be returned.
- Example: if(tourCompany is valid) enter trial name, time, day, month and year.
- Example output: <valid company, wild path, 1400, 15, march, 2021>.

createReservation(int, int, bool, bool): null
<40, 5, True, True, tour.reservationDetails>
tour.reservationDetails = [40, 5, True, True]

createNewEvent(String, String, String): String[ ]
<“Ice Cream Day”, “October 27th, 2021”, “9:00am”, event.eventDetails>
event.eventDetails = [“Ice Cream Day”, “October 27th, 2021”, “1:00pm”]

addTrailDetails(int, varchar): null
<7, “Kilimanjaro National Park”, trail.trailDetails>
trail.trailDetails = [7, “Kilimanjaro National Park”]

cancelEvents(): null
<emergencySituation, event.eventDetails>
event.eventDetails = [“Ice Cream Day”, “October 29th, 2021”, “1:00pm”]

cancelTours(): null
<emergencySituation, tour.reservationDetails>
tour.reservationDetails = [0, 0, False, False]

closeTrails(): null
<emergencySituation, trail.trailDetails>
trail.trailDetails = [0, “Kilimanjaro National Park”]

closePark(): null
<emergencySituation, park.parkOpeningHours, park.parkClosingHours>
park.parkOpeningHours = [0, 0, 0, 0, 0, 0, 0]
park.parkClosingHours = [0, 0, 0, 0, 0, 0, 0]

5.3.2 System Test #2

This system test will utilize 10 different functions in order to portray usage of the entire system. First, a user will be created that is an admin in order to display the changes that can be made by someone who has direct access to the data within the system. Next, one aspect of each of the three pages will be updated: a trail's ID and location, a new event, and updating the availability of the reservation list. After those three aspects have been updated, the user will be able to click on the tour page, the local information page, and the trail page in order to view the updated information that has been made.

createUser(String, String, Bool): null
<"park.manager", "parkmanager1234", True, user.username, user.password, user.isAdmin>
user.username = "park.manager"
user.password = "parkmanager1234"
user.isAdmin = True

addTrailDetails(int, varchar): null
<10, "Kilimanjaro National Park", trail.trailDetails>
trail.trailDetails = [10, "Kilimanjaro National Park"]

createNewEvent(String, String, String): String[ ]
<"Free Reservations Day", "October 20th, 2021", "9:00am", event.eventDetails>
event.eventDetails = ["Free Reservations Day", "October 20th, 2021", "9:00am"]

createReservation(Int, Int, Bool, Bool): null
<50, 10, True, True, tour.reservationDetails>
tour.reservationDetails = [50, 10, True, True]

isTourPageClicked(): null
displayReservation(): object[ ]
<tourPageClicked, displayReservation()>
<True, [50, 10, True, True]>

isLocalPageClicked(): null
displayEvent(): String[ ]
<localPageClicked, displayReservation()>
<True, ["Free Reservations Day", "October 20th, 2021", "9:00am"]>

isTrailPageClicked(): null
displayTrailDetails(): object[ ]
<trailPageClicked, displayReservation()>
<True, [10, "Kilimanjaro National Park"]>

6.0 Changes Made To The UML Class Diagram

6.1 Changes Made For System Tests

There have been a few changes made to the UML Class Diagram in order to match the test cases we have provided. The following attributes have been added: “tourDay” in the “updateTour()” function, “tourMonth” in the “updateTour()” function, “tourYear” in the “updateTour()” function, “eventDetails” in the “Local Events” class, and “reservationDetails” in the “Reservation” class. These attributes were added so that our test cases could have variables within them that would store the data we needed. For example, “eventDetails” was created to contain the name, date, and time of an event happening, all in one variable. The following functions have been added: “createUser()” in the class “userClass”, “createReservation()” in the class “Reservation”, “displayReservation()” in the class “Reservation”, “tourInformation()” in the class “Tour Information”, “tourGuides()” in the class “Tourist Guides”, “companyProfile()” in the class Company Profile, “getRating()” in the class “getRating”, “cancelTours()” in the “checkForEmergencySituation()” function, and “closeTrails()” in the “checkForEmergencySituation()” function. These functions were added so that our test cases could accurately show how each attribute within the class was getting updated. For example, the “tourInformation()” function was created in order to be able to update all the variables within the “Tour Information” class. The following modification have been made to certain functions: the “addTrailDetails” function now takes the parameters of type int and type varchar, the “updateSafety()” function now takes parameters of type int, double, double, and bool, the “updateInfo()” function now takes parameters of type string and type int, and the “updateTour()” function now takes parameters of type string, string, string, int, int, int, and int. These modifications were made so that some functions that didn’t take any parameters before, now took parameters and could be used to update certain variables. For example, “addTrailDetails” previously took no parameters, but now it takes an int and a varchar in order to update “trailID” and “location” respectively.

6.2 Changes Made For Databases

With the addition of our databases, the UML Class Diagram has altered slightly. Now, there are two databases (the Park Database and the User Database) that each connect to the specified classes that they affect. Firstly, the Park Database connects to the following classes: “Company Profile”, “Trail”, “Local Events”, “Reservation”, and “Home Page.” The reason the Park Database connects to all five of these classes is because it contains five databases within it, with each database containing information used by one of the five classes. The following databases, in order, correspond to each of the five classes listed: the Company Database, the Trail Database, the Local Database, the Availability Database, and the Weather Database. Secondly, the User Database is exclusively composed of attributes related to the user, so the class that would use the data from this database is the class “userClass.”

7.0 Database Diagram

For our Kili Trekker System, the best implementation of databases would be to have two separate databases, with one being SQL and the other being NoSQL, particularly document based. Our SQL database will be for the Park Database, and it will contain several databases within it, including the Company Database, the Trail Database, the Local Database, the Availability Database, and the Weather Database. SQL will be best suited for these databases because the content within the databases was able to be categorized very nicely and reduce redundancy for our databases. For example, in the Trail Database, many of its attributes are associated with other databases, allowing for it to access more information than it directly stores. Each database shown in the Park Database will have a one to one relationship, as each database will only correspond to one unique value of the corresponding database. Our NoSQL database, the user database, will contain information related to the accounts users will create on the website. The reason we felt a document based noSQL database would be best suited for this is because some of the data for our users would be better stored on a document, where it could list attributes such as job description and tour history.

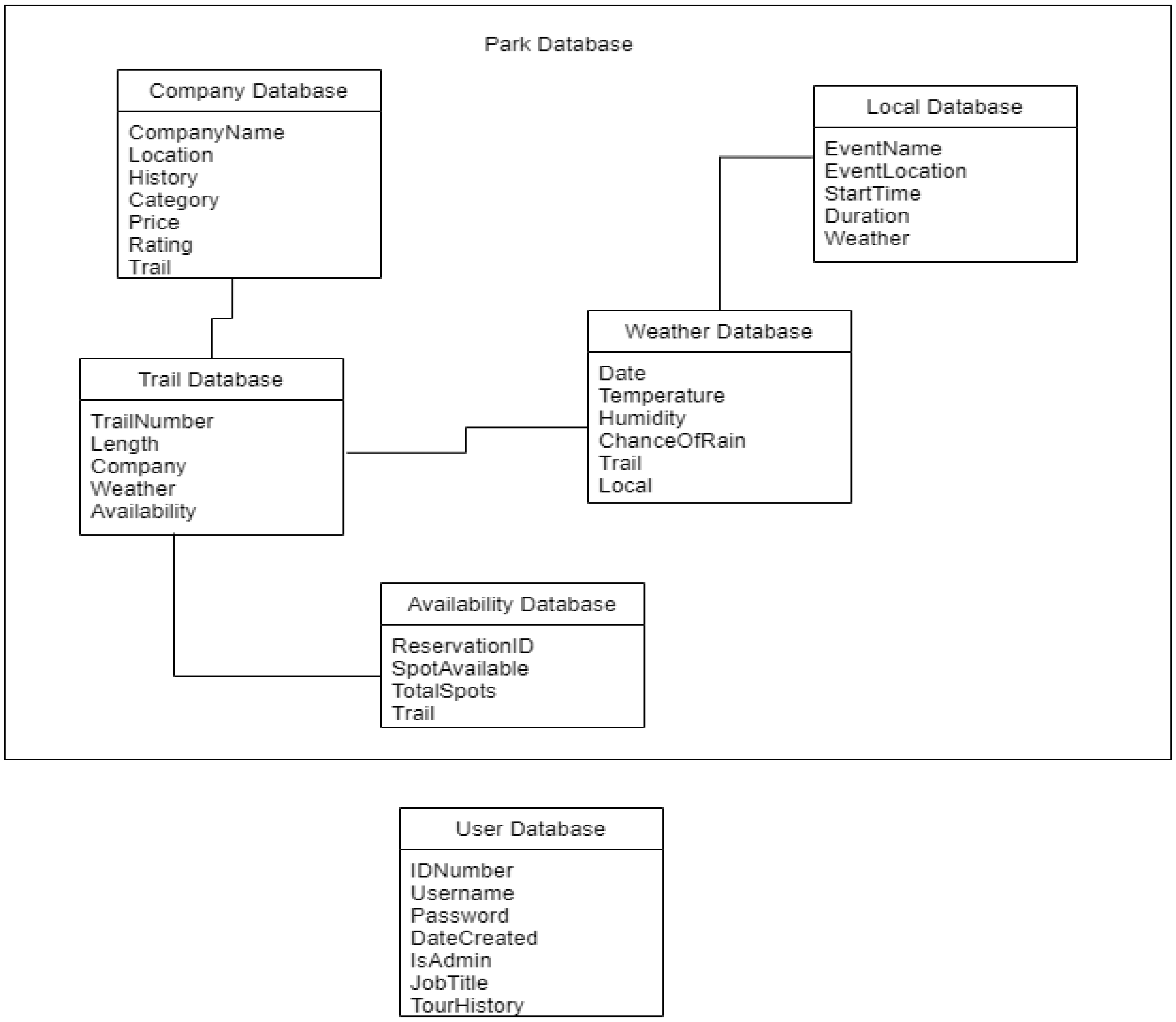

Figure 7.0.1 Database Diagram

8.0 Data Dictionary

Database 1.0 - Company Database

Name: Company
Type: Table
Description: This database includes information about the companies that are allowed to provide services for any type of activity in the area.
Primary index: CompanyName

Name: CompanyName
Type: Attribute/Alphanumeric
Description: This will include the name of the guide company in order to have a unique identifier for each company.
Expected length: 5 - 50 characters

Name: Location
Type: Attribute/Alphanumeric
Description: This will include the location where the company will be located in case of any circumstances or problems that might need to be resolved.
Expected length: 5 - 25 characters

Name: History
Type: Attribute/Integer
Description: This will keep track of the number of times the company is requested to provide services.
Range of values: 0 - 100

Name: Category
Type: Attribute/Alphanumeric
Description: The purpose of this attribute is to categorize the company in 3 categories: whether the company offers services for local events, trails or both.
Range of characters: 5 - 25 characters

Name: Price
Type: Attribute/Float
Description: This attribute will store the different prices for each company based on the quality of the service and the type of the event.
Range of values: 500.00 - 3500.00

Name: Rating
Type Attribute/Float
Description: This will provide the rating range that customers input in order to provide

feedback for the company and help future customers to decide which company is the best option for them.
Range of values: 0.00 - 5.00

Name: Trail
Type: Attribute/Integer
Description: This will provide the number of the trail that this guide company will be leading, allowing the company database to pull information from the trail database about the specified trail.
Range of Values: 1 - 12

Database 1.1 - Trail Database

Name: Trail
Type: Table
Description: Contains information about the 12 different trails within the Kilimanjaro National Park.
Primary Index: TrailID

Name: TrailNumber
Type: Attribute/Integer
Description: Contains a number that will correspond to one of the 12 trails.
Range of Values: 1 - 12

Name: Length
Type: Attribute/Integer
Description: Describes the length of the trail in kilometers.
Range of Values: 0 - 100

Name: Company
Type: Attribute/Alphanumeric
Description: Contains the name of the company that is associated with this trail, linking the trail database with the company database.
Expected Length: 5 - 50 characters

Name: Weather
Type: Attribute/Alphanumeric
Description: Contains the date, which will be used as a link between the trail and the weather database in order to gather information on what the weather will be like on the specified date for the specified trail.
Expected Length: 8 characters

Name: Availability
Type: Attribute/Integer

Description: Contains a unique 4-digit ID related to the reservations for the specified trail number, linking the availability and the trail databases to determine the total spots available for the trail.
Range of Values: 0000-9999

Database 1.2 - Local Database

Name: Local
Type: Table
Description: Contains information related to the events happening in the local area. This includes the name of the event, the location of the event, and the duration of the event.
Primary Index: EventName

Name: EventName
Type: Attribute/Alphanumeric
Description: This will describe the name of the event, and will be used as the primary index to distinctly identify each event, as every event will be named differently.
Expected Length: 5 - 20 characters

Name: EventLocation
Type: Attribute/Alphanumeric
Description: This will describe the location of the event, such as where in the park the event will take place, and whether or not the event will take place at one of the trails.
Expected Length: 5 - 20 characters

Name: StartTime
Type: Attribute/Alphanumeric
Description: This will describe the starting time of the event in the format of “00:00”, ranging from values starting at “00:00” and ending at “23:59”.
Expected Length: 5 characters

Name: Duration
Type: Attribute/Integer
Description: This will describe the duration of the event by containing the amount of hours the event will last, with the max amount of time being “24”, symbolizing that the event will last all day.
Range of Values: 0 - 24

Name: Weather
Type: Attribute/Alphanumeric
Description: This will contain the date that the event is taking place on, allowing the local database to access the weather database in order to see the weather for that day. If there is bad weather on the specified date, then the event will be rescheduled.
Expected Length: 8 characters

Database 1.3 - Availability Database

Name: Availability
Type: Table
Description: Contains information pertaining to the availability of spots for a certain trail, such as the trail number, the number of spots available, the total number of spots, and the guide that will be assigned to the specified trail.
Primary Index: ReservationID

Name: ReservationID
Type: Attribute/Integer
Description: This will contain a unique 4-digit ID that will associate with a reservation for a particular trail number.
Range of Values: 0000 - 9999

Name: SpotsAvailable
Type: Attribute/Integer
Description: This will describe the number of spots open for reservation for a specified trail, ranging from 0 spots available, to the maximum number of spots, which is 500.
Range of Values: 0 - 500

Name: TotalSpots
Type: Attribute/Integer
Description: This will describe the total number of spots that can be reserved for a specified trail, ranging from the minimum of 100 spots, to the maximum of 500 spots.
Range of Values: 100 - 500

Name: Trail
Type: Attribute/Integer
Description: This will describe the trail number that is associated with the specified reservation, allowing for the availability database to access the trail database.
Range of Values: 1 - 12

Database 1.4 - Weather Database

Name: Weather
Type: Table
Description: Contains information about the weather, such as date, the temperature in celsius, the humidity percentage, the chance of rain, the trail number, and the event ID.
Primary Index: Date

Name: Date
Type: Attribute/Alphanumeric

Description: This will be the primary index of the weather database, as the date will always be unique since there will never be two of the exact same day with the exact same year. This attribute will be formatted to look like "MM/DD/YY".
Expected Length: 8 characters

Name: Temperature
Type: Attribute/Integer
Description: This will describe the temperature recorded for the specified date in celsius, with 0 degrees celsius being the minimum since it corresponds with 32 degrees fahrenheit, and 50 degrees celsius being the maximum since it corresponds with 122 degrees fahrenheit.
Range of Values: 0 - 50

Name: Humidity
Type: Attribute/Integer
Description: This will describe the humidity percentage for the specified date, with the percentage being a number between 0 and 100.
Range of Values: 0 - 100

Name: Precipitation
Type: Attribute/Integer
Description: This will describe the percentage of precipitation for the specified date, with the percentage being a number between 0 and 100.
Range of Values: 0 - 100

Name: Trail
Type: Attribute/Integer
Description: This will describe the trail that this weather will be present in. Since the trails are in different areas of the park, there might be different weather conditions depending on which trail is taken, so each trail number will be recorded in the weather database, linking the two databases.
Range of Values: 1 - 12

Name: Local
Type: Attribute/Alphanumeric
Description: This will contain the name of the event that will be held on the specified date. If there is no event being held that day, the name will be "NULL."
Expected Length: 8 characters

Database 2.0 - User Database

Name: User
Type: Table
Description: Contains information about the users that are inside of the database. These users will have basic information about them, a tour history, and will contain permissions on administrative access.
Primary Index: IDNumber
Secondary Index: Username

Name: IDNumber
Type: Attribute / Integer / Primary Index
Description: A unique ID that is generated for each user. The user will be unable to see this ID, and it is solely used to reference them by either administrators or directly through the database.
Range of Values: 0 - 2, 147, 483, 647

Name: Username
Type: Attribute / Alphanumeric
Description: The username that our users will log in with to access their account information
Expected Length: 5 - 20 characters

Name: Password
Type: Attribute / String / Password / Alphanumeric / Symbols
Description: The password for our users, should include one uppercase letter, one symbol, and one number.
Required Length: 5 - 30 characters

Name: DateCreated
Type: Attribute/ Integer / Constant
Description: The date the account was created, this should be initialized upon account creation and should remain constant until the account is deleted from the database.
Format: DDMMYYYY
Required Length: 8

Name: IsAdmin
Type: Attribute / Boolean
Description: Indicated administrator privileges. If this is true, then the user has access to the administrator home page, and can access the database directly.
Initial Condition: False

Name: JobTitle
Type: Attribute / String / Alphanumeric

Description: Description of the job that people perform. This can indicate a tour guide, or an internal administrator, and certain permissions will be locked behind these titles. Not required for all users.
Expected Length: 1 - 30 characters

Name: TourHistory
Type: Attribute / Document
Description: Provides a tour history for the user to download. This will store the date, time, receipt, and link to the website of the tour company.
File Size: < 1mb

9.0 Security

This section of the document will show potential security risks, and the countermeasures that we have incorporated in our system to prevent them.

9.0.1 Backdoor - Integrity

The risk of a user logging in as an admin when they aren't an admin is a security risk we have identified that attacks the integrity of our login system. In order to prevent this, we will be using two-factor authentication in order for users to log in. The two-factor authentication process will use an app that users can download on their mobile devices, and once they sign in once on their device, they can select the option to stay signed to skip the two-factor authentication when they login again from that same device.

9.0.2 Denial-of-service - Availability

A potential threat to the availability of our website is a denial of service attack on our host server. In order to combat this threat, we are implementing a VPN on our host server in order to mask the ip of our server.

9.0.3 Direct-access attacks - Availability

Directly accessing the host server would potentially cause the servers to shut down, or unauthorized access to an administrator account. In order to protect against this, we recommend that the building the server is in is locked, and security check the area frequently.

9.0.4 Eavesdropping - Confidentiality

The risk of unauthorized access to listen to communication between our server and the user's device is a risk that endangers the confidentiality of our system. In order to combat this, we will have antivirus software installed on our server that will be specifically focused on dealing with eavesdropping software. If it identifies one, then it will immediately handle it so that no information would be gained from the eavesdropping software.

9.0.5 Phishing - Confidentiality

There are two main vulnerabilities from phishing attacks that we have identified. The first vulnerability that was identified was the potential fake link that a tour guide company may post. To solve this, the park supervisor will manually review each link before the tour is posted. The second vulnerability lies within false emails sent by people attempting to scam the user out of their payment info. To combat this, we will never use email as a line of communication besides verifying their address, and this will be displayed when creating an account.

9.0.6 Bot Detection - Integrity

The risk of bots leaving fake ratings for companies in order to boost their star rating is a security risk we have identified that attacks the integrity of our rating system. We will combat this risk by only allowing users to write a rating for the guide company after having fully completed a trail. This will be tracked by allowing users who have signed up with certain trails to have the option for a review unlocked after the estimated time duration of the trail has been completed.

9.0.7 Privilege escalation - Integrity

The situation where a user is able to elevate their permissions to that of an admin is a risk that could be detrimental to the integrity of our website. If a user were to gain the permissions of an admin, they would be able to view and alter any data they wanted on the website. Therefore, in order to combat this, admins must input their unique employee ID number they were given when they were hired every time they wish to view or edit data stored within the website. If a user somehow manages to elevate their privileges to that of an admin, they will be able to do nothing different than a regular user, as they will not have access to the unique employee ID that the admins are physically carrying on them.

10.0 Life-Cycle Model

In order to complete this project, we utilized a slight modified version of the waterfall model. We tackled each process individually, but with slight iteration. First, we gathered the system requirements from the user in order to know what we would be building. All of these requirements and their functionality were listed in the system description in section 2. Next, we took these requirements and created our design specifications, which would be used in order to build the software system. Then, we tested our design with our implementation in order to verify that our system worked. We used three unit tests, two integration tests, and two system tests to prove our system functioned as intended. In order for these tests to flow smoothly, we had to modify our design and UML class diagram to match all the functions we were calling in our black-box tests. Afterwards, we created our two databases that would be used by the system to gather and store data. Initially, our UML class diagram did not include a database, so we went back and slightly modified our design specifications and class diagram to match the implementation of our databases. Lastly, we created a discussion of several security risk countermeasures once our system's designs were both validated and verified. We believe this approach to the project covered everything necessary and was appropriate for this project due to the scale of the design being asked. However, if the project were to have been scaled up, this kind of approach would have resulted in a lot of backtracking. Since we focused on each part individually, if one change was created to any of the previous steps, then we would have to go back and change it. Therefore, at a larger scale, the changes could become massive, resulting in a lot of confusion and time lost. If we were to improve upon this approach in the future, we would most likely follow an approach similar to the spiral model, where iteration happens at each step to make sure that certain design elements are finalized before moving on, saving time in the process.