San Diego State University

Software Design:
Kili Trekker System

Version: 2.0

Members: Aatusa Mehdiyan, Jenny Nguyen, Minh Nguyen, Jose Payan

GROUP 13

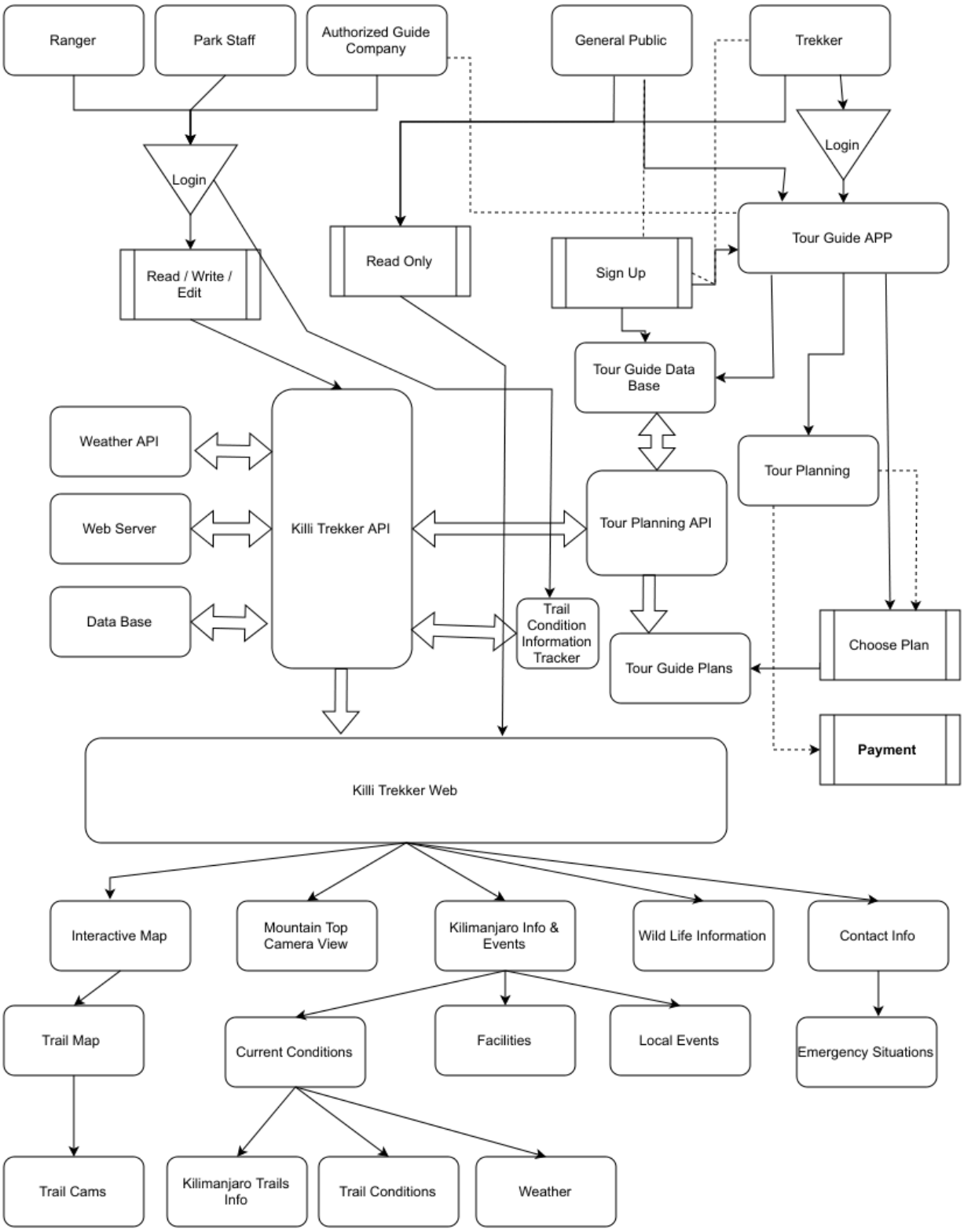
CS 250

Professor Bryan Donyanavard

November 30, 2021

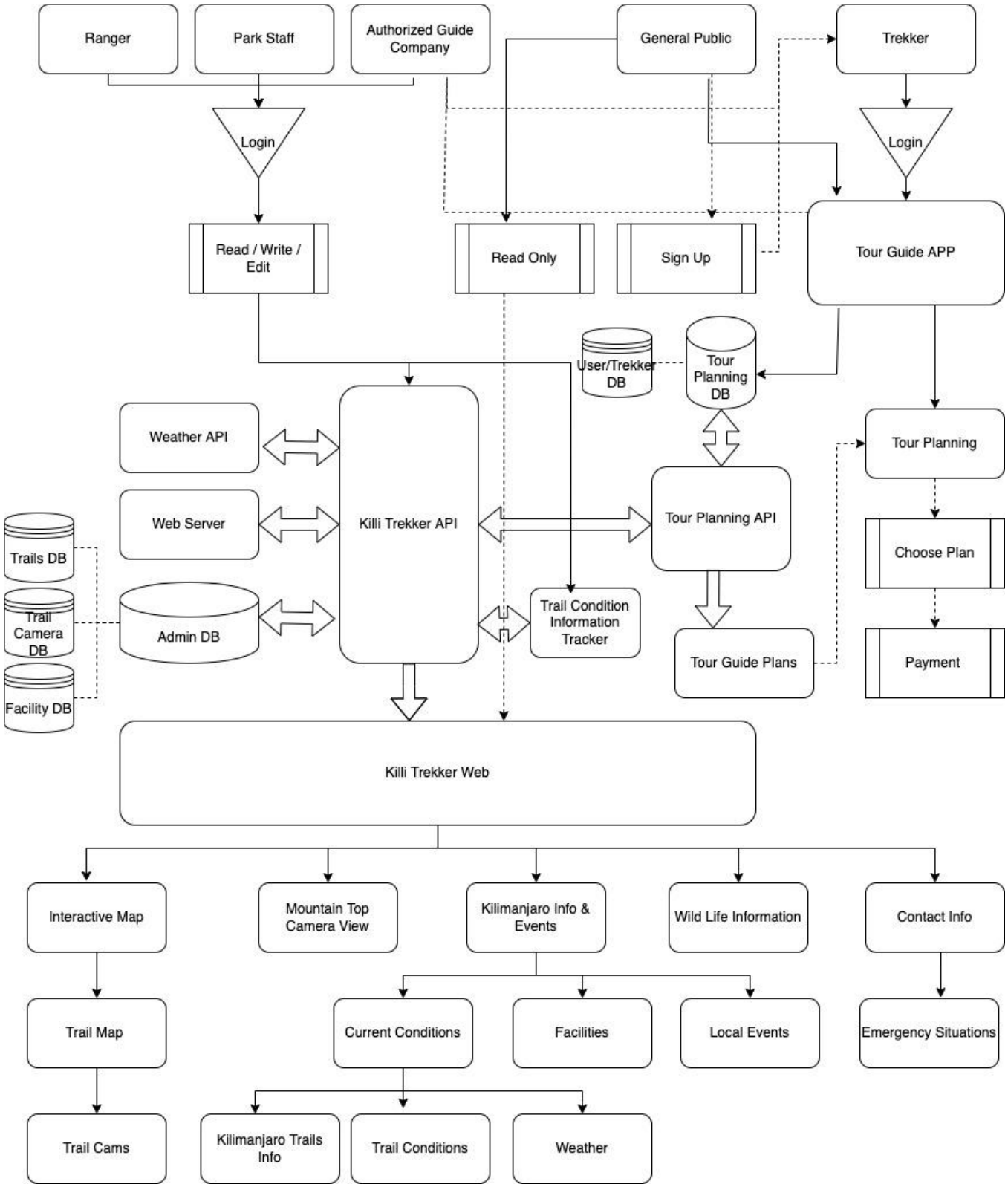
Software Design Specification 1.0:

- Architectural diagram of all major components

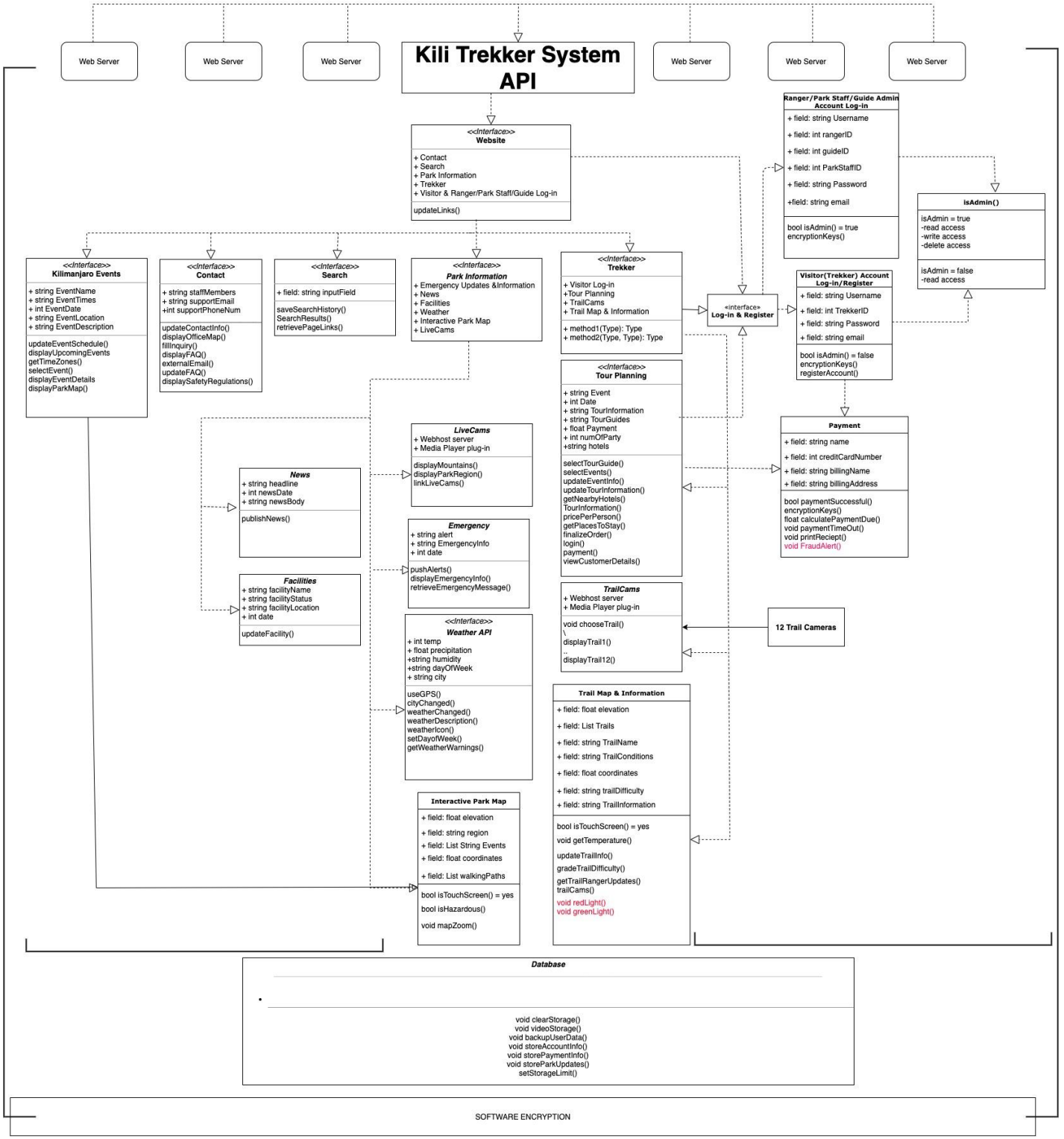


Software Design Specification 2.0:

- Architectural diagram of all major components including databases



- UML class diagram



Description of classes, attributes, and operations of Kili Trekker System:

isAdmin: This class is responsible for running, checking and managing secure logins from users who want to access the system. If the output is TRUE, they can have full access as read, write and delete to the system. If the output is FALSE, they only have read access.

- ➔ Attributes: bool isAdmin || *reads in accounts IDs. Determines whether to give read access, write access, delete access to user, returns true or false*
- ➔ Operations: bool isAdmin = false || *read access*
  bool isAdmin = true || *gives user read, write, and delete access*

Ranger/Park Staff/Guide Admin Account Log-in: This class stores encrypted account information of the system administrators. Each admin will have a different username, ID, password and email used to log into the system. If it is TRUE, the class runs to isAdmin().

- ➔ Attributes: string Username, int rangerID, int guideID, int ParkStaffID, string Password, string email
- ➔ Operations: isAdmin() = true
  -Void encryptionKeys() || *operation takes in user information details and returns an encrypted hash of data to the database to store*

Visitor(Trekker) Account Log-in: This class stores the encrypted account information of park visitors. This class also holds the function to allow first time users to register an account to the website.

- ➔ Attributes: string Username, int TrekkerID, string Password, string email
- ➔ Operations: bool isAdmin() = false
  -Void registerAccount() || *registration request method if first time user*
  -Void encryptionKeys() || *operation takes in user information details and returns an encrypted hash of data to the database to store*

Payment: This class is used by the TourPlanning class. Sends user to a safe payment processing page and takes in their payment information

- ➔ Attributes: int creditCardNumber, string name, string billingName, string billingAddress, float amountDue
- ➔ Operations: bool paymentSuccessful() || *checks for valid card information and returns true or false*
  -Float calculatePaymentDue() || *reads in user's tour planning order, calculates and returns paymentDue*
  -Void paymentTimeout() || *goes to timeout page and redirects user to original order page when they do not submit payment within time frame or payment method fails*
  -Void encryptionKeys() || *operation takes in user payment details and returns an encrypted hash of data to the database to store payment history*
  -void printReceipt() || *prints a receipt page of order and sends customers a copy of their receipt to their email address.*

<<interface>>

Log-in: command class for access between user and system

Website: Upon successful login, it leads to the main information directory which is located entirely within the website to support users**.**

- ➔ Attributes: contact, search, park information, trekker, visitor and ranger/park staff/guide log-in.
- ➔ Operations: Links(), pageRedirect()

Kilimanjaro Events: It will contain the main information about the events of each trail including the name, time, date and location of the event taking place. It will be updated by Admins and presented on the system in the menu of Interactive Park Map.

- ➔ Attributes: string EventName, int EventTimes, intEventDate, string EventLocation, string EventDescription

➔ Operations: updateEventSchedule() || *method for park staff to update and publish event schedule*
-displayUpcomingEvents() || *retrieves updates of events and displays upcoming events to website*
-void getTimeZones() || *gets user time, displays clock to page + countdown to event*
-void selectEvent() || *method for user to choose event and return a link to the event page*
-void displayEventDetails() || *function to print specified event details including event name, time, date and description*
-void displayParkMap() || *displays park's interactive map to event location*

Contact: It contains contact information of staff in the area including email and phone number so that everyone can easily connect and support when needed.
➔ Attributes: string staffMembers, string supportEmail, int supportPhoneNum
➔ Operations: updateContactInfo() || *The system shall add new contact info*
-void displayOfficeMap() || *The system shall display the office map*
-void fillInquiry() ||*method option for user to directly send an inquiry on website page*
-void displayFAQ() || *displays updated FAQ strings to page*
-void externalEmail() || *directs user to an external email window after clicking on Kilimanjaro Park support email*
-displaySafetyRegulations() || *function for park staff to edit and publish park safety regulations to page*

Search: The function that helps the user to find all the necessary information of the park area that is displayed in the system.
➔ Attributes: string inputField, list searchResults
➔ Operations: void saveSearchHistory() || *takes in user inputs, saves and displays users search history to bottom of search bar*
-void SearchResults() || *takes in user's inputField string, searches database with keywords to search relevant pages*
-void RetrievePageLinks() || *prints list searchResults and gives links to pages relevant from searchResultsMethod().*

Park Information: This page contains links for the user to view and interact with the website's major features
➔ Attributes: LiveCams, Interactive Park Map, Emergency, News, Facilities, WeatherAPI
➔ Operations: getWeather() || *connects page to weather API to display Kilimanjaro weather conditions*

Emergency: Any emergency situation taking place in the park will be updated and alarms by admins that are displayed in the Park Information directory.
➔ Attributes: string alert, string EmergencyInfo, int date
➔ Operations: pushAlerts || *sends alerts to all rangers, park staff, guides, and park visitors to mobile devices and desktop*
-displayParkRegion() || *prints screenshot of map where emergency is located*
-displayEmergencyInfo() || *prints emergency information to page*
-retrieveEmergencyMessage() || *collects rangers' emergency information and sends it to main server for staff to update to system*

News: This function allows read access for visitors to view latest and previous park news that are published by park staff who are granted write permissions to edit news information.
➔ Attributes: string headline, int newsDate, string newsBody
➔ Operations: PublishNews() || *method to edit, add, and publish city and park news article*

Facilities: This function allows users to know the construction of new facilities or the condition of the park's facilities.

➔ Attributes: string facilityName, string facilityStatus, string facilityLocation
➔ Operations: updateFacility() || *add, delete, edit and publish park facility information*

LiveCams: The live camera that observes and reports the weather data on the mountain top in the park area which is displayed in the Park Information section.
➔ Attributes: Web Host server, Media Player plug-ins
➔ Operations: displayMountains() ||
-linkLiveCams() || *establishes connection between main server to park cameras*
-displayMountains() || *switches cameras to mountain view*
-displayParkRegion() || *switches cameras to park region view*

Weather API: This widget gathers latest weather information from the internet and displays forecasts to the website. It is user interactable so that they are able to view weather conditions on a specified day.
➔ Attributes: int temp, int precipitation, string humidity, string dayOfWeek, string weatherWarning
➔ Operations: useGPS() || *uses GPS mod to generate accurate precipitation predictions, returns precipitation*
-weatherChanged() || *detects significant change in weather, updates weather status immediately to API*
-DisplayWeatherDescription() || *collects weather data from selected dayOfWeek, returns temp and weather predictions*
-weatherIcon() || *displays rain, sunny, cloudy icons*
-setDayofWeek() || *prints display of weather of the week*
-getWeatherWarnings() || *calculator the temperature and humidity, grades them and returns weatherWarning status*

Interactive Park Map: Uses maps API to display park content and imagery for display on web pages and mobile devices. It features four basic map types (roadmap, satellite, hybrid, and terrain).
➔ Attributes: float elevation, string region, List String Events, float coordinates, List walkingPaths
➔ Operation: bool isTouchScreen() = yes || *allows user to interactive map with touch display devices*
-void mapZoom() || *takes touch and mouse input to return map zoom*
-void isHazardous() || *If there are hazardous conditions within the park reported by the staff, the system will update this information.*

Trekker: This class is geared towards the visitor allowing them to read, select, plan or print all the information about trails, tours, information and map of the park.
➔ Attributes: Visitor Log-in , Tour Planning, TrailCams, Trail Map & Information
➔ Operations:method1(Type) + method2(Type, Type)

Tour Planning: It contains reading information about the park tour when trekkers log into the system it helps them to plan a trip based on this information.
➔ Attributes: Event, Date, TourInformation, TourGuide, float paymentDue, numOfParty, hotels
➔ Operations:
-login() || *redirects user to login page before continuing through tour planning process*
-selectTourGuide() || *prints park tour guide option menu. User selects a tour guide and function returns tour guide name and saves it to tour planning order*
-selectEvents() || *prints list of events and menu options for user to view event information relevant to tour date*
-updateEventInfo() ||*edit, remove, and publish event information*
-getNearbyHotels() || *get Kilimanjaro nearby hotels and display booking choices for for visitor*
-TourInformation() || *edit, update, and publish tour information by admin*
-float pricePerPerson() || *takes in number of party members touring and calculate pricing before returning price to payment processing page*

-finalizeOrder() ||*takes user to payment processing page to finalize order*
-payment() || *system interacts with payment class*
-viewCustomerDetails() || *method for tour guide company to store and access customer information*

TrailCams: displays live cam of the trail conditions and allows visitors and guides to select a specific trail camera to view

- ➔ Attributes: Web Host server, Media Player plug-in
- ➔ Operations: chooseTrail() || *prints and takes input from trail menu with options to view specific camera*
  -void displayTrail1()....displayTrail12() || *changes camera view to specified trail number*

Trail Map & Information: TrailMap class displays an interactive map to users and allows visitors and guides to gain access to trail information. Admin users have write permissions to edit trail information.

- ➔ Attributes: List string Trails, string TrailName, string TrailConditions, string coordinates, string trailDifficulty, string TrailInformation
- ➔ Operations: isTouchScreen() = yes || *Allows users to access and interact with trail map through mobile and other touch screen devices*
  -void gradeTrailConditions() || *interacts with weather API, takes in conditions of trail by ranger and grades the trail conditions whether it is safe or not to go to*
  -void gradeTrailDifficulty || *saves and stores a map of easy, medium, to hard levels of difficulty to list of trails*
  -void updateTrailInfo() || *edit, remove, and publish trail information by admin*
  -void getTrailRangerUpdates() || *retrieves status updates discovered by ranger, sends it to main server for park staff to update to system*
  trailsCams() || *interacts with TrailCams class to display camera views to page*

Database: holds video storage, backups user data and stores account information. Also contains edit website history and update history for a certain period of time.

- ➔ Operations: clearStorage() || *cleans up unnecessary files and functioned called when storage limit is hit*
  -backupUserData() || *makes a copy of user data*
  -storeAccountInfo() || *stores registered accounts to the website*
  -storePaymentInfo() || *saves an encrypted history of payment transactions*
  -storeParkUpdates() || *saves a yearly report with date, year, and time of updates to the park website's history*
  -setStorageLimit() || *storage holds data up to a certain amount of days*

Testing Goals

System Acceptance Testing: Complete System

- This testing will be performed to determine if the Kilitrekker System has met the requirement specifications and also evaluate the overall system's compliance with the park company's requirements. Another significant goal of this testing is to verify if it has met the required criteria for proper delivery to end users including current/potential park visitors, rangers, park staff, and tour guides. New acceptance tests should be conducted for each iteration during development for quality assurance and to confirm correctness.

Function/Integration Testing: Interfaces among integrated units

- This document will overview the testing of the multiple components of the Killi Trekker System and view if the components function together properly. We will analyze how the external systems interact with the main system and check for proper data flow and database changes between modules. Thorough testing of the interfaces will help formulate a successful user experience when using the Kilitrekker website.
- **Functional Testing:**
    - Testing the software in its entirety to confirm that each function, feature or module of the trekker is working as intended.
- **Integration Testing:**
    - Testing multiple modules after integrating them together to detect potential defects. Trekkers must communicate with different modules. We will be checking for performance, data flow and any changes that take place during their interactions.

Unit Testing: Single Code "unit" (e.g., method)

- **User Interface Test Cases:**
    - Check functionality of textboxes, buttons, dropdowns, internal and external links, and page scaling.
- **Input Data Validation:**
    - Input data validation will encompass a range of checks that may be adopted generally to the data which is entered into an application system. It will check and establish:
        - Mandatory Fields testing
        - Unique Field Values testing
        - Null value testing
        - Acceptance of allowable characters
        - Negative values testing
        - Field length specifications
        - Improbable Value testing
        - Garbage Value testing

Trekker Class

System Acceptance Testing: Complete System

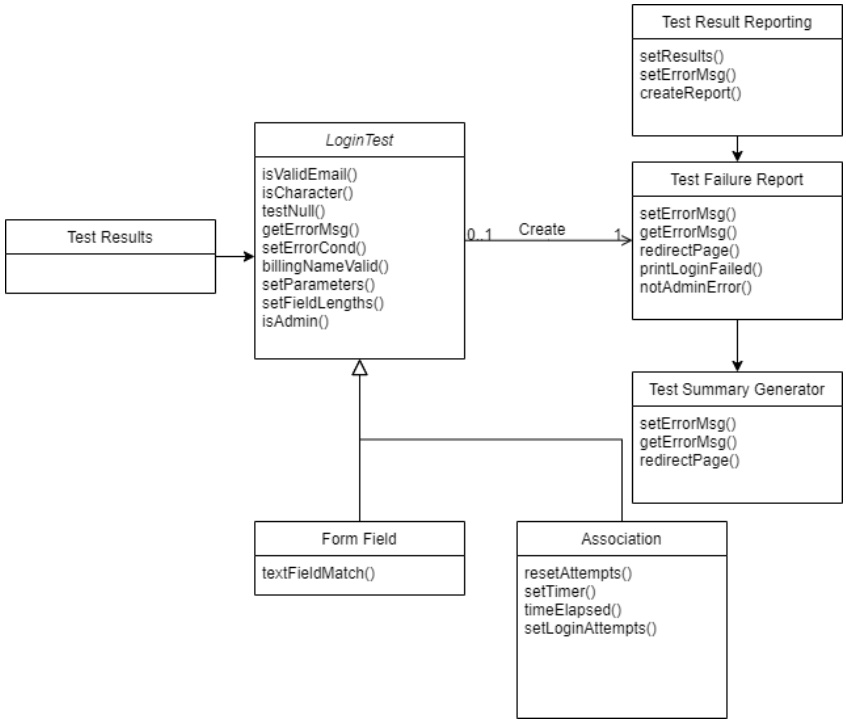
Trekker Class contains the following main components:

- LoginManager
- TourPlanning
- TrailInformation
- Payment

Trekker
LoginManager
TourPlanning
Payment
TrailInformation

System/acceptance testing:
Login
Plan a tour
View trail information

LoginManager



System Acceptance Testing: Test objective for the LoginManager is to allow users to have their account information and settings stored into the system efficiently. They will be able to access and update their personal information to their preference and also view their Tour information/history. LoginManager module controls are:

- Login
- Logout
- Register

Function/Integration Testing:

- The Login Manager module will store the following information into the Killitrekker system's database after successful login. This module will communicate with the database, and the overall Killitrekker website. Login information in particular will connect with the TourPlanning and Payment modules to allow users to complete their tour bookings.
    - Int: rangerID
    - Int: parkStaffID
    - Int: guideID
    - Int:trekkerID
    - String: username
    - String: password
    - String: email
    - Int: loginDate
- The expected fields are checked.
- A welcome message must appear after successful login and an error message must appear on the screen after an invalid login
- There must be stored site cookies for login fields.
- "Forgot Password"("Reset Password")/ "Forgot Username"/ "ForgotID" functionality options should be present
- Login attempts: disable for one hour after 3 unsuccessful login attempts
- Successful login will bring users to the Welcome page, Home page, or to the original page they were on.

Unit Testing:

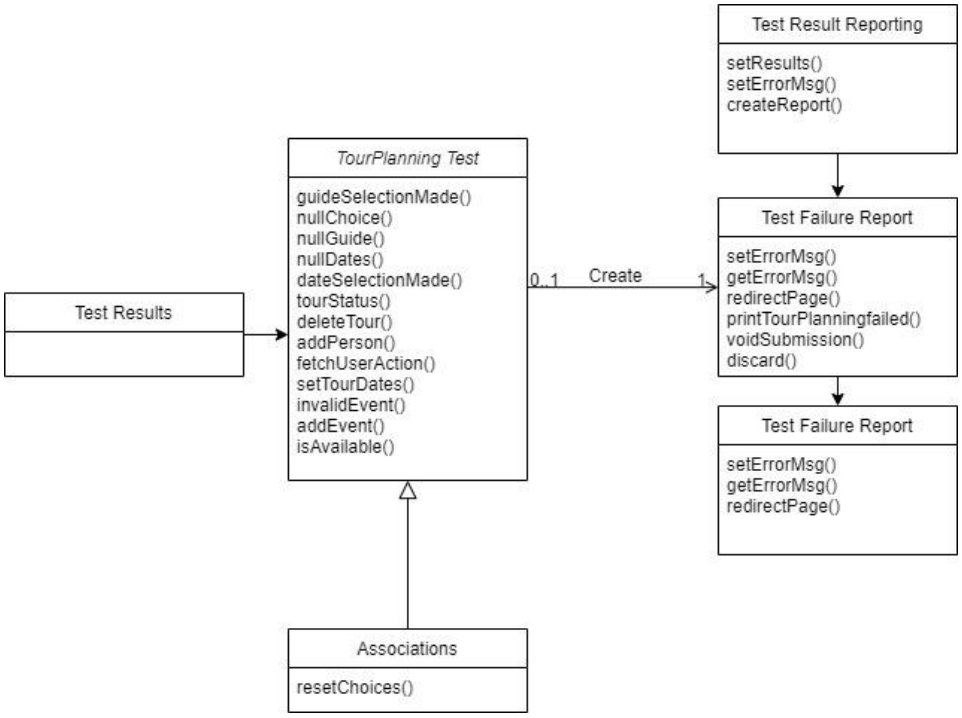
- **Input data validation:** In order for users to successfully log-in, they must satisfy the required fields in order to further access the system's other major components for security. Such testing will check for secure and reliable user interface and usability. Specifications include:
    - Valid E-mail address must exist
    - Passwords are case sensitive and must be alphanumeric. Length should be between 8 to 32 characters.
    - Text fields cannot exceed the signified lengths
    - Errors for Null inputs in email addresses including empty inputs
    - No Invalid characters

	isAdminLogin	MyAccount
Login String: Email String: Password int: ID bool: isAdmin void setIsAdmin(...)	LoginTest bool {true, false}: loginTest if (loginTest=F) return readAccess; else if (loginTest=T) return fullAccess; else return 0; Web: connectedWeb void readAccess(...) void fullAccess(...)	Login SettingsManager TourPlan Payment RegisterAccount
Unit testing: Access.setIsAdmin(...)	Function/Integration testing: LoginTest.readAccess(Login)	System/acceptance testing: Access to website Read all park information Login Logout

Test Case #	Test Case Description	Test Steps/Data:	Expected Results	Actual Result	Pass/ Fail
1	User goes to the website and clicks the Register button to sign up. User types in their email, UserID, and password in the empty fields and click submit. Check response when valid information is entered	Email: user123@email.com UserID: 34567889 Password: pass999	Registration should be unsuccessful. Users must enter a valid password with a symbol for highest security. Users should not log in and the website will display an error page and require the user to try again.	Registration unsuccessful	Fail
2	Ranger goes to the website and clicks the Login/Register button to sign in. Ranger types in their email, rangerID, and password in the empty fields and click submit. Check response when valid information is entered	Email: ranger99@email.com RangerID: 09123442 Password: lNf9^Oti7^2h	Login should be successful and grant the ranger read/write permissions to the system.	Login successful	Pass
3	Check when required fields are not filled and no data is entered.	E-mail: ID:	Page should display a mandatory symbol(*)	Registration unsuccessful	Fail

		Password: Click-> Register	next to required fields in red.		
4	Check when the user inputs an invalid e-mail for log-in.	E-mail: eee@hotmail.com ID: 67896654 Password: Morty2018!	Page will refresh and highlight the “Email” field and note to the user, “Email does not exist within our system. Please try again. ⅔ login attempts left”	Login unsuccessful	Pass
5	Check when the user inputs invalid characters into text fields.	E-mail: yakuzadestroyer22@yahoo.com UserID: aasdklasdk Password: KiryuDameDane1&	Page will refresh and highlight the “UserID:” field. A red message will appear and tell the user, “Invalid characters. Please only enter numeric values only”	Login unsuccessful	Pass

TourPlanning



System Acceptance Testing: Testing objective of the TourPlanning module is to ensure users are able to book their tours at the park without issue. Their payment information, tour dates, guide

information, party size, and events will be customizable and displayed for them when they access their profile. Specified tour guides will be allowed to see the park visitor's selected info necessary for tour guide's planning preparation.

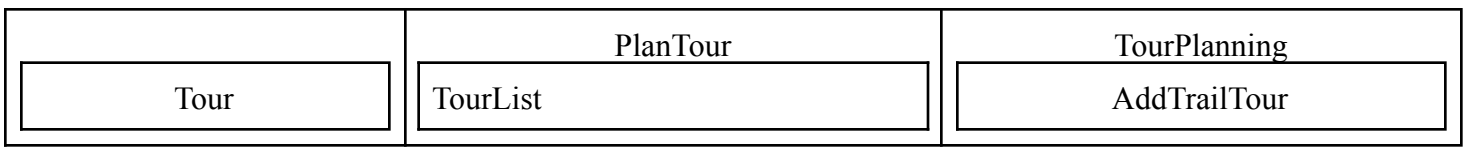
- Plan tour
- Choose tour dates
- Choose a trail to tour
- Add event
- Add tour guide
- Add person to party
- Pay for tour

Function/Integration Testing: TourPlanning module will have multiple components sending data to each other and will be stored personally into park visitors' accounts. Testing will ensure that user actions are fetched and selections load and update the information from other pages with no issue. TourPlanning connects to the Payment module to calculate payment amount for the tours and process payments successfully.

- List<TourGuides>: string guides
- List<TourInformation>: string tourInfo
- string: tourGuideName
- String: tourGuideAbout
- Bool: isAvailable
- Void chooseGuide(...)
- Void chooseDate(...)
- Void addPerson(...)
- Void addEvent(...)

Unit Testing:

- Links and booking modules are tested. This unit testing relies on users to interact with the components directly by clicking buttons and using dropdown menus to select specific values. Testing will oversee:
    - Successful button triggers on all devices such as desktop and mobile(func fetchUserAction(...))
    - UI buttons are displayed properly to user and accurately labeling
    - Ease of selection of tour dates through a pop-up calendar widget that allows users to pick the days they will be visiting
    - Clear use of menu choices of tour guides, events, and trails
    - Max people to add to the tour party is 30

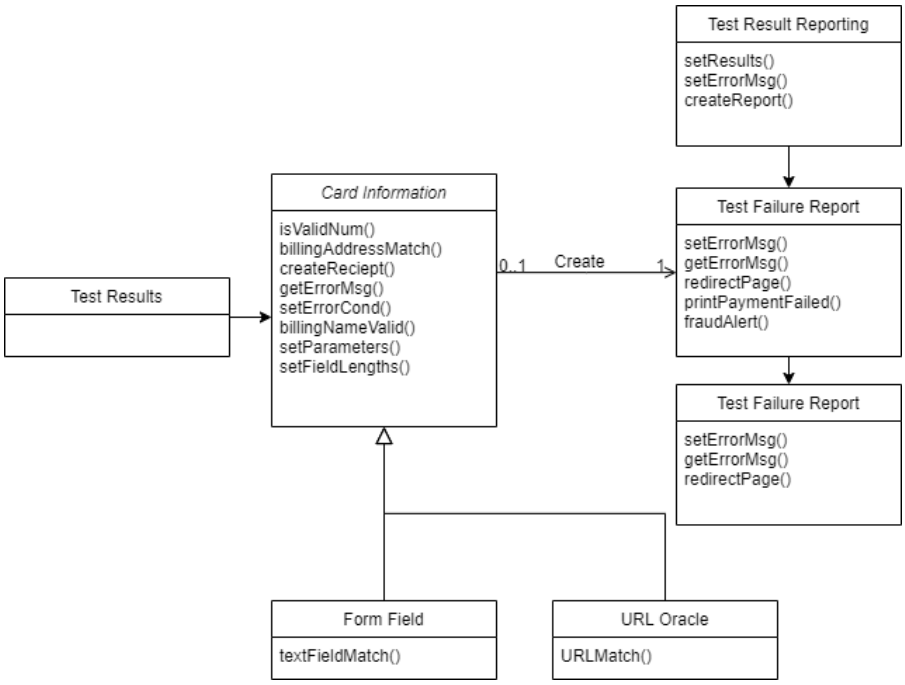


String: tourName Int: TourDates	List<TourGuides> tourGuideNamesList App: connectTour	TourDates
Void setTourDates(...)	void addEvent(...) void deletedTour(...)	AddEvent
Unit testing:	Function/Integration testing:	ChooseGuide
Tour.setTourDates(...)	PlanTour.addTour(Tour n)	AddPerson
		TourPayment
		Refund
		System/acceptance testing:
		Add a Trail Tour Delete tour dates Remove event Remove tour guide Remove person from party Refund for tour

Test Case #	Test Case Description	Test Steps/Data	Expected Results	Actual Result	Pass/Fail
1	User interface testing: Check functionality of trail condition buttons, links, trail dropdown options, and textboxes.	Click on the dropdown menu to select a trail. Ex) User selects and clicks the "Trail 1 Conditions" link.	Page redirects to Trail 1 information and displays Trail 1's camera, condition rating, and difficulty.	Page redirect and Trail 1 media display successful.	Pass
2	User interface testing: Check functionality of unavailable trails.	Click on the dropdown menu to select a trail. Ex) User selects and clicks the " Trail 7 Conditions" link.	Page redirects to Trail 7's camera, condition rating, and difficulty.	Page redirect and displays " Trail 7 Condition: Poor , Tour Unavailabl e".	Fail
3	User interface testing: Check when user does not hit a trail	User does not click on a trail from the drop down menu.	Page does not redirect to another page. Page will highlight the user to	Page does not redirect and stays on the	Fail

	selection		select a trail from the dropdown menu.	original page.	
4	User interface testing: When a user adds a person to a party, the price of the tour updates. Users are able to change the amount of people by clicking the plus or minus buttons next to the quantity of people field.	User hits the plus sign next to the person quantity field to add someone to the tour.	Page updates immediately and party of the tour updates to 2.	Person added successfully. Price of the tour updates	Pass
5	Check when user does not select tour dates or a tour guide.	User leaves tour dates unselected from the calendar widget. User does not click on a tour guide name from the drop down menu and hits submit.	Page updates immediately to notify the user must enter a tour date and select or tour guide to continue their tour order Void these choices from being entered into the system.	Tour order not confirmed	Fail

Payment



System Acceptance Testing: Testing objective of the Payment system component is to test that the payment processor appointed by a merchant can handle transactions from different channels. It will check and receive details to respective issuing banks or associations. It will also successfully accept payment information from users, perform and record transactions safely, and send such information to relevant parties.

- Select payment method
- Enter payment details
- Store payment details

Function/Integration Testing: In order for park visitors to complete their tour booking, the Payment component communicates with the TourPlanning component which will send payment funds to the Kilitrekker company bank account and also save and record transaction history details to the database. Payment details will also be externally sent out by email to customers.

- String: name
- String: billingAddress
- Int: cardNumber
- Int: securityCode
- Int: date
- Float: amountDue
- Void addPayment(...)
- Float calculatePaymentDue(...)
- Void paymentType(...)
- Bool paymentSuccessful(...)
- Void paymentDetails(...)

- Void refund(...)

Unit Testing:

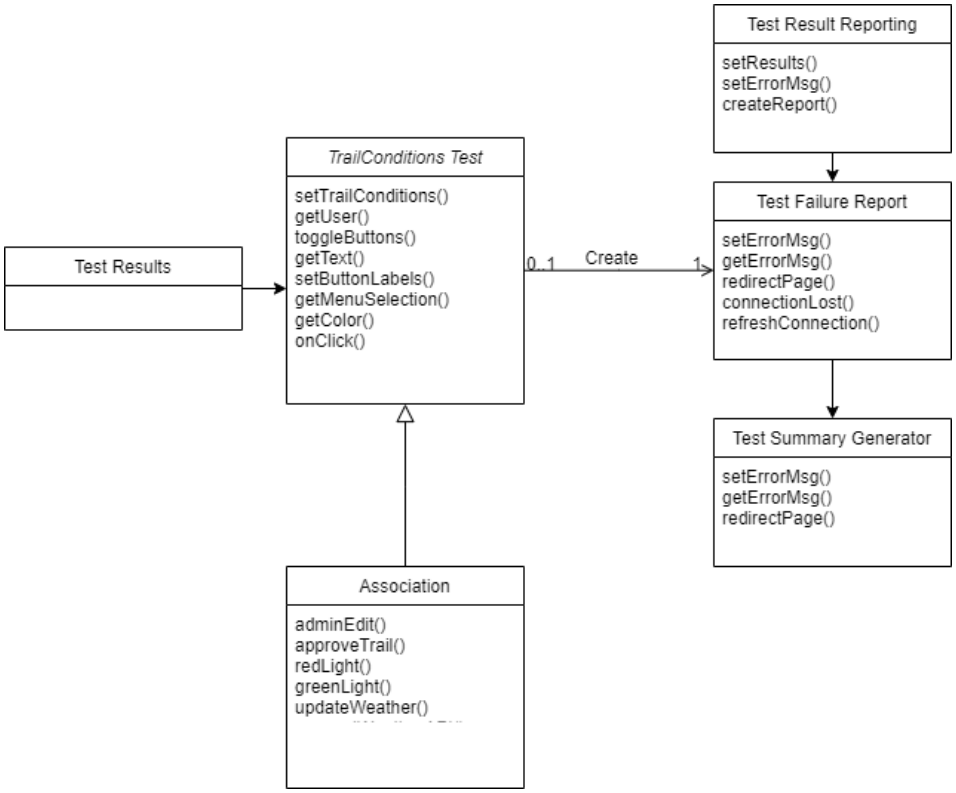
- **Input data validation:** In order for users to successfully book and pay for their tour, they must satisfy the required payment fields and security measures. Specifications include:
  - Card information matches with billing and identity information
  - Card number field should be limited to 16 digits, no alphabetical letters or symbols are enterable.
  - CV pins are required
  - Card information is up to date and valid
  - No Invalid characters in integer accepting fields including null inputs
  - Fields cannot exceed the signified lengths

	PaymentProcess	MyPayment
Card	CardTest	SelectPaymentMethod
String: cardName String: cardAddress int: cardNum int: cardCodeNum int: cardDate float: Payment	bool cardTest if (cardTest = false) return 0; else return 1;	EnterPaymentDetails
void setPayment	void approvedCard(...) void errorCard(...)	CancelPayment
		PrintPayment
		ViewPayment
		LoginManager
Unit testing: Card.setPayment(...)	Function/Integration testing: PaymentProcess.approvedCard(Card)	System/acceptance testing: Cancel a payment to get a refund View a payment of finalized order Print a receipt of payment Login logout

Test Case #	Test Case Description	Test Steps/Data	Expected Results	Actual Result	Pass/Fail
1	To pay for their order tour of services. They need to click the “Checkout” button, and enter shipping and	Billing Name: John Smith Billing Address: 2312 Marlesta St. CA 92111 Card number: 2342 5234 4356 2345 CV: 345 Expiration: 02/23	Authorization to approve with correct card information input and successful message display	Payment successful	Pass

	payment details to the required fields. Then they click the “Submit" button to get a verified transaction.		on the payment page.		
2	To pay for their order tour of services. They need to click the “Checkout” button, and enter shipping and payment details to the required fields. But, they miss CV and wrong card numbers. Then they click the “Submit" button to get a verified transaction.	Billing Name: Jasmine Nguyen Billing Address: 2435 Convoy St. CA 92121 Card number: 1256 124 4566 120 CV: ??? Expiration: 12/21	Authorization to decline with incorrect card information input and failure message display. User will be directed back to the payment page to try again.	Payment unsuccessful	Fail
3	To pay for their order tour of services. They need to click the “Checkout” button, and enter shipping and payment details to the required fields. But they delay entering the required fields process too long. The transaction request is not verified.	Billing Name: …. Billing Address: ... Card number: 2323 5234 000 000 CV:... Expiration: 11/25 Payment process declares remaining time out on the page. Click the “yes" button to continue. Click “no” or run out of time countdown to end.	Authorization to decline and the session expired message display. Users will be directed back to the payment page to try again.	Payment unsuccessful	Fail

TrailI Condition InformationTracker



System Acceptance Testing: Test objective for the Trail Condition Tracker is to allow real time results of the trail conditions. The Tracker should efficiently update current weather conditions pulling information from the Killi Trek Website. A menu will display all trails and direct to the appropriate page for further evaluation. The Killi Trekker API handles requests made from the Trail Condition Tracker. All push notifications are displayed with valid input results.

- Approve Trail Condition
- Unapproved Trail Condition

Function/Integration Testing: The Trail Condition Tracker will confirm availability for safe trails for the Tour Planning Application. The application will be able to update real time with current trail conditions by accessing the Weather Application Interface database and Killitrekker website. The weather condition will be pushed from Weather API to the KilliTrekkerAPI in accordance with requests. The data will be converted in the Trail Condition Information Tracker determining appropriate criteria for an end result.

- View Trail Information
- List<TrailList>: trailList
- List<trailInformation> string trailInfo
- String : trailName “n”
- Void selectTrail(…)

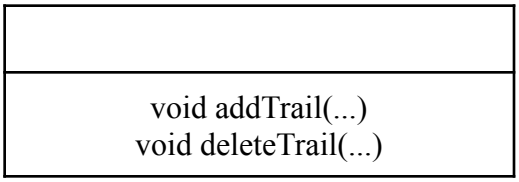
- Void selectTrailHistory (. . .)
- View Trail Conditions
- Select a Trail Camera

Unit Testing:

- **Real Time Update of Trail Information :** For approval of trail availability , satisfied conditions must be met. As well as approved adminID for manual approval for moderate conditions. All requirements must be met with proper permission and a final request for confidence of the final result.

    - Select a Trail : null Trails will not be accepted
    - Current Weather : “Good, Moderate, Warning: Risk” Real Time Weather update from the weather API ; Good: clear weather conditions ; Moderate: High or Low Temperature ,
    - View Camera : Select Type of Weather and Forecast
    - Approve Trail : input adminID and Password
    - Choose Green/ Red : Default : “Auto” (takes parameters “Good; Moderate; Warning” from KilliTreekerAPI to approve or mark trail unavailable. Green approves ; Red marks trail as unavailable.

	TrailConditionTracker	Trail Information
Trail	TrailList	Choose trail
String : trailName bool : isSafe	List<Trails> : trailList App: connectTrailCam AP : connectKiliTrekkerAPI selectTrail(n) Void getCurrentWeather(string “date”)	View trail information
void setIsSafe(...)	string viewCamera(currentTemp, trailLiveCam int n) Approve trail : enter adminID and Password Choose Green/Red : (default : Auto ; GreenLight : isSafe ; else “RedLight: isNotSafe” NOTE : click green or red on Manual Override	View trail conditions
		Select a trail camera
Unit testing: Trail.setIsSafe(...)		System/acceptance testing: Approve Trail Mark Trail as Unavailable Select Trail Weather Type Select

	 Function/Integration testing: TrailConditionTracker.addTrail(Trail n)	

Test Case #	Test Case Description	Test Steps/Data	Expected Results	Actual Result	Pass/Fail
1	This tests the trails for their condition and approval or denial for tour guide requirements. Accesses the database and updates the approved trails and information in real time and matches weather conditions. All good conditions are approved.	Select Trail : Trail 7 Current Weather : "Good" View Camera : Sunny , calm weather Approve Trail : "adminID" Green/Red : Green	Display message "String : Green : isSafe"	Green Light "This Trail Conditions are Safe "	Pass
2	This tests the trails for their condition and approval or denial for tour guide requirements. Accesses the database and updates the trails as unavailable and information in real time and matches weather conditions. All "Warning: Risky" conditions	Select trail: " Trail 5" Current Weather : " Warning : Risk " View Camera : Raining , Windy Approve Trail : "Enter adminID" Green/Red : Red	Int 0 , Display message " String : Red : isNotSafe"	Red Light "This Trail Conditions are not Safe"	Fail

	are unapproved.				
3	This test approves an Administrators manual input to override moderate trail conditions with a valid adminID and Password. All invalid IDs and Passwords will not be able to make changes.	Select Trail : Trail 3 Current Weather : “ Moderate: Risk ” View Camera : “Cold , Windy” Approve Trail : “hack1234 , password: hacking” Green/Red : Red	Invalid ID	Error invalidID, Please enter correct ID and Password.	Error
4	Last Verification Step Method for Manual Approval. Ensures confidence in final review.	Select Trail from List : Trail 7 Current Weather : “ Moderate: Risk ” View Camera : “High Temp , Dry” Approve Trail : “admin1232 , password: passWord1232” Green/Red : Red	Are you sure you want to approve trail: “Yes Approve, No Set As Unapproved	adminID: Approved Trail Thanks for Reviewing Trail 7	Pass
5	This test objectively test the Green/Red default set to “Auto” where the Trail Information Tracker automatically approves any trail with good weather conditions. Set trail as isSafe when weather conditions are “good” or moderate only.	Select Trail: 3 Current Weather : “Good” View Camera : “ 73 Degrees, Sunny” Approve Trail : “admin1232 , password: passWord1232” Green/Red : Auto: Green	Auto: Green Current Weather: “Good”	Auto : Trail Approved	Pass
6	This test objectively test the Green/Red default set to “Auto” where the Trail Information	Select Trail: 2 Current Weather : “Moderate : Risk” View Camera : “ 22 Degrees, Windy” Approve Trail : “admin1232 , password: passWord1232” Green/Red : Auto: Red	Auto: Red Current Weather: “Moderate”; ”Warning”	Auto : Trail Unapproved	Fail

	Tracker automatically marks a trail and sets it as unavailable. Set trail as isNotSafe when weather conditions are “Moderate” and “Warning: risky”.				

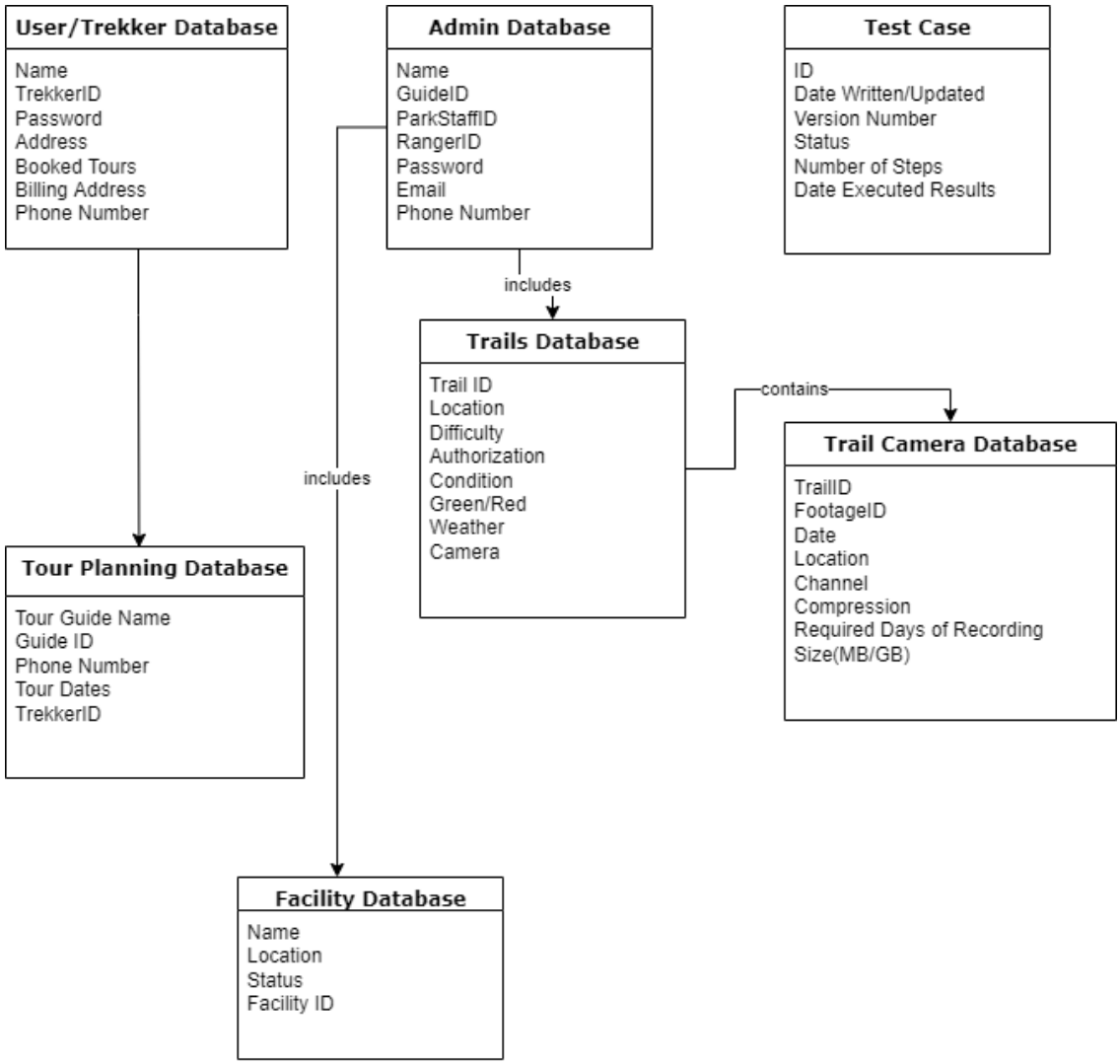
Noted Changes and Updates:

- Anti-Fraud measures/alerts against suspicious card transactions will be considered and integrated into the payment processor. <<void FraudAlert(...)>>
- Reorganization of the Trekker Trail component to update and show trail conditions and information at once and onto the same page.
- Automated Color-coded range grading of trail conditions that are consistently updated as data is retrieved from and analyzed by cameras, ranger/staff reports, and weather conditions <<void redLight(...)>>, <<void greenLight(...)>>

SQL Data Management Strategy:

- Killi Trekker System will be utilizing five SQL databases to store the following data: User/Trekker Data, Admin Data, Trails Data, Tour Planning Data, and Trail Camera Data. User and Admin Login data will collect information of the complete system's account information of Trekkers, rangers, park staff, and tour guides. Tour Planning database is contained by the User/Trekker class and will collect tour information reports and store private information to the user. Booked tour information and history will also be stored there including dates and guide information. Facility database is included in the Admin database as it can be only modified with administrative privileges. The Facility database will store facility identification numbers, logs, location, and status updates. The Trails database will store trail log and condition reports and the trail camera database will store footage information and footage storage limits. Trails database will collect trail camera database information as well. Each database specifies accepting data types, inputs, and outputs.

System Database Diagram



Data Dictionary:

Data Dictionary Admin DB:
Name: Person
Type: Table
Description: Kili Trikker employee
Primary Index: GuideID, ParkStaffID, RangerID
Secondary Index: Name

Name: Email
Type: Table
Description: Administrator's email
Expected values: Attribute: Alphanumeric/Integer
Optional: No
Length max: 30

Name: GuideID
Type: Attribute/Integer
Description: Kili Trikker guide employee ID
Length Min: 4
Length Max: 7
Range of values: 0-9999999

Name: ParkStaffID
Type: Attribute/Integer
Description: Kili Trikker park staff employee ID
Length Min: 4
Length Max: 7
Range of values: 0-9999999

Name: RangerID
Type: Attribute/Integer
Description: Kili Trikker ranger employee ID
Length Min: 4
Length Max: 7
Range of values: 0-9999999

Type: Phone Number
Description: User Phone Number
Expected Length: 11
Range of values: 0-9999999

Data Dictionary User/Trekker DB:
Name: Person
Type: Table
Description: Kili Trikker visitors
Primary Index: TrekkerID
Secondary Index: Name

Name: TrekkerID
Type: Attribute/Integer
Description: Kili Trikker ID
Length Min: 4
Length Max: 7

Name: Email
Type: Table
Description: Trekker's email
Expected values: Attribute: Alphanumeric/Integer
Optional: No
Length Max: 30

Type: Phone Number
Description: User Phone Number
Expected Length: 11
Range of values: 0-9999999

Data Dictionary Tour Planning DB:
Name: Tour
Type: Table
Description: Information of Kili tour
Primary Index: Tour Guide Name
Second Index: Guide ID, Trekker ID

Type: Guide ID

Data Dictionary Trails DB:
Name: Trail ID
Type: Attribute/Integer
Description: Kili Trikker trail ID
Length Min: 4
Length Max: 7
Range of values: 0-9999999

Description: Kili Guide ID Range of values: 0-9999999 Type: Trekker ID Description: Kili Trikker ID Range of values: 0-9999999 Type: Phone Number Description: User Phone Number Expected Length: 11 Range of values: 0-9999999 Type: Tour Dates Description: Trekker Tour Dates Booked Primary Index: Start Date Secondary Index: End Date Expected Length: 8 Optional: No	Name: Location Type: Table Description: Trail location info Primary Index: ID Secondary Index: Address Name: Difficulty Type: Attribute/Alphanumeric Description: Trail Difficulty Grading Primary Output: Integer range 1-10 Secondary Output: String range “Easy”-”Moderate”-”Difficult” Name: Authorization Type: Attribute/Alphanumeric Description: Approve Trail with adminID and Password Range of values: 0-9999999 Output: String: “Green”/”Red” Name: Condition Type: Attribute/Alphanumeric Description: Condition grading of trails Primary Index: TrailID Range of values: 0-9999999 Name: Camera Type: Attribute/Alphanumeric Description: Trail Camera Media Live Feed Primary Index: Camera ID, Trail ID, Trail Name Range of values: 0-9999999 Media Playback: MP4
Data Dictionary Trail Camera DB: Name: TrailID Type: Attribute/Integer Description: Kili Trekker Trail ID Length Min: 4 Length Max: 7 Range of values: 0-9999999 Name: FootageID Type: Attribute/Integer Description: Kili Trekker camera footage ID Range of values: 0-9999999 Name: Locations Type: Table Description: Trail camera location info	**Data Dictionary Facility DB:** Name: FacilityID Type: Attribute/Integer Description: Kili Trikker facility ID Range of values: 0-9999999 Name: Facility Type: Table Description: Facility name Primary Index: Name Name: Location Type: Table Description: Facility location info Primary Index: ID Secondary Index: Address

Primary Index: ID Secondary Index: Address Name: Compression Type: Attribute/Alphanumeric/Integer Description: Kili Trikker camera compression Range of values: 20 MB - 15 GB Expected: MP4 Type: Recording Dates Description: Required Days of Recording Primary Index: Start Date Secondary Index: End Date Expected Length: 8 Optional: No Type: Channel Description: Channel IP Connection to Main Office Primary Index: IPv4/IPv6 address Expected Length: 128 bits Example: 2001:0db8:85a3:0000:0000:8a2e:0370:7334 Range of Values: 0 - 3.4 x 10^38 Optional: No	Name: Status Type: Attribute: Boolean True/False Description: Facility in Construction