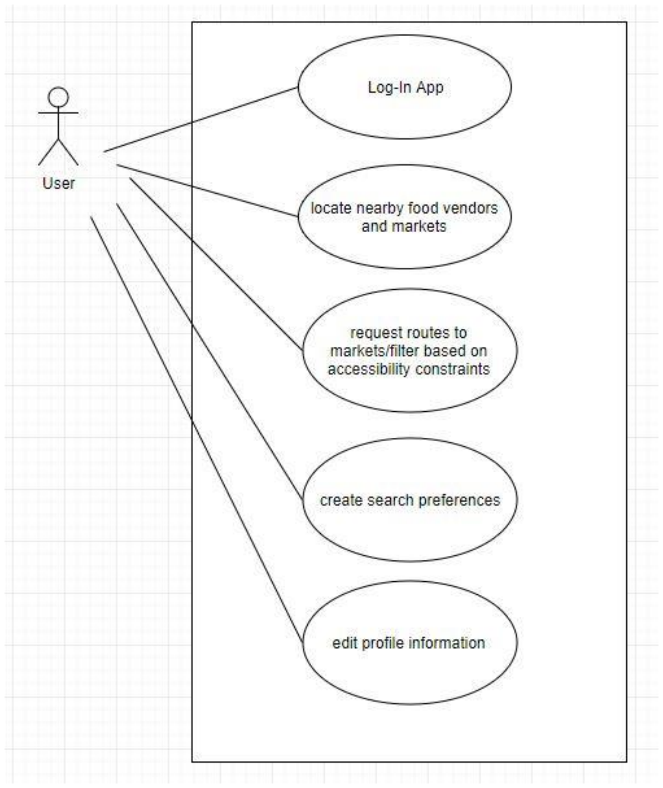
Amauri Lopez

CSC415-01 Assignment 5

November 10, 2017

OSS Analysis and Design

Use Case Descriptions + UML Use Case Diagram:

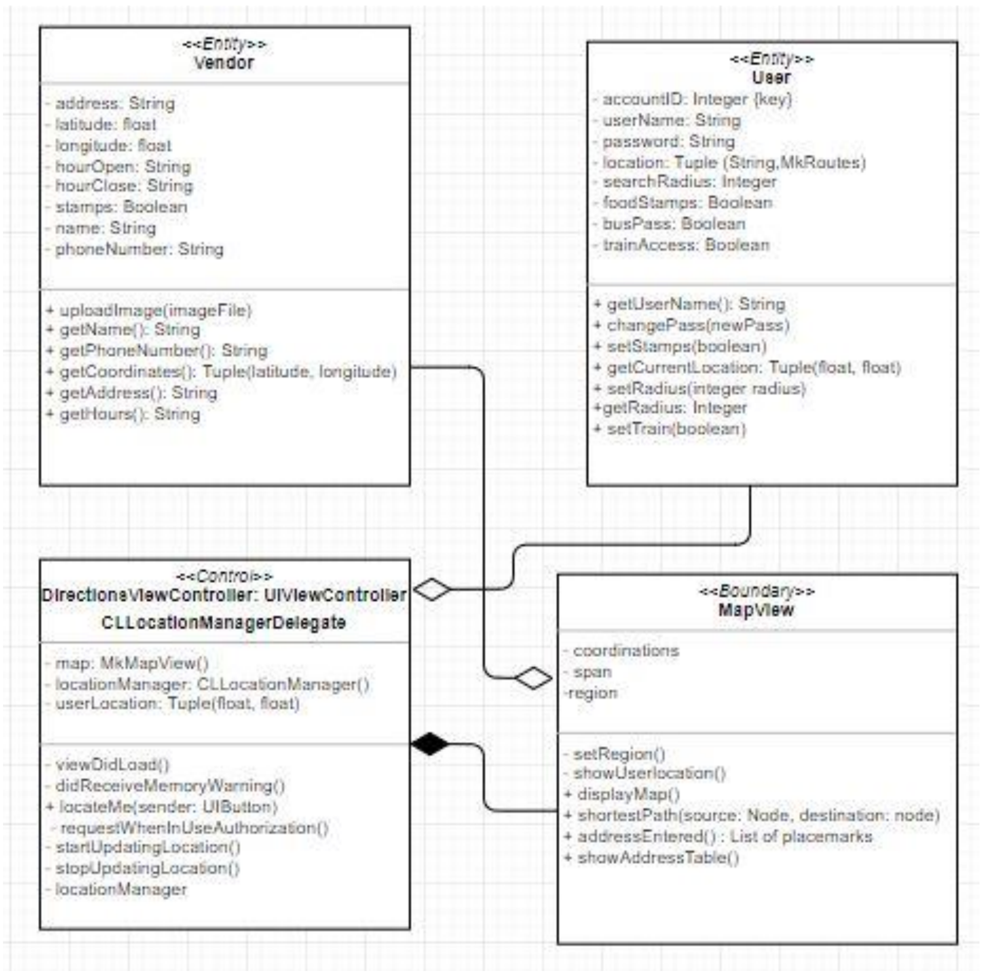

Log-In App
User
locate nearby food vendors and markets
request routes to markets/filter based on accessibility constraints
create search preferences
edit profile information

- **Log-In**: Because each user will have user specific information (i.e. do you have a car, do you have access to the bus/ is there a train station in your area/ do you have access to this stop/ do you only walk). This information will be different for every user, therefore a log-in will be implemented to designate between users -> will create the "User" object from the User class and assign these attributes upon account creation.
- **Request routes/read routes filtered by accessibility constraints:** The App is supposed to help improve access to food markets & vendors around the area. It will do this by providing routes to such places, which will change depending on the user's accessibility constraints specified through their user profile information (i. e. if they can drive, then driving routes will be the default routes displayed. If they can only take the bus then a combination of bus + walking routes will be shown). These routes will try to focus on distance/transportation as well as the time it will take to get there
- **Create search preferences:** The user should be able to set search/route preferences such as maximum distance/time willing to travel, or how large the search radius would be when location new food vendors/markets.
- **Edit profile information**: The first UI page that the user will be presented with is their profile page. The user should be able to specify accessibility constraints here (if any) as well as customize search preferences here as well. Personal information like picture or home address or things like that should not be needed.

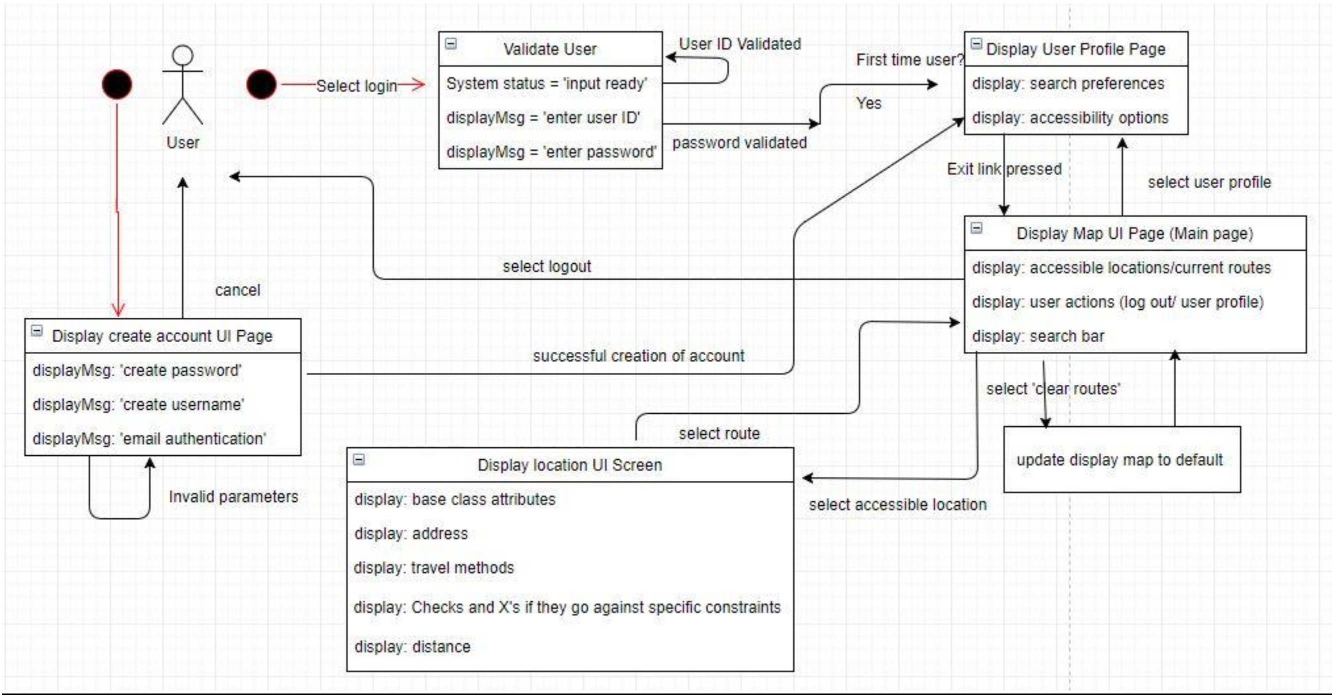
Design Class Diagram + Description:

- **Vendor Class:** Food Vendor places such as food markets/soup kitchens/possibly churches/grocery stores/possibly restaurants are classified under a food "Vendor" base class, which would contain it's address and geolocation (latitude + longitude) as well as hours of service and such. Thus, food markets & other "Types" of vendors should probably inherit from this class and exhibit different attributes like "acceptStamps" Boolean and "img" attributes for their particular image displays on the map. Maybe add "List variables in these classes such that each vendor can be organized in separate lists, which may all be part of a greater "Vendor" list.

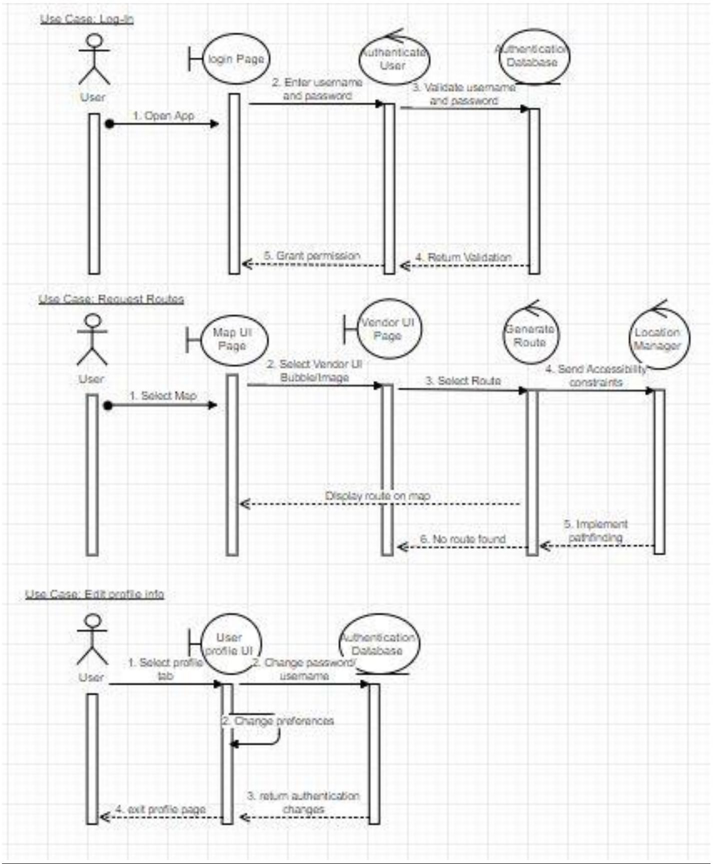
- **User/Account Class:** For obvious reasons, we should have a user profile class to hold user profile data. Should be tied to an overall "Account " class which has their username/password as well as their accessibility options (location, search radius, user food stamps, bus pass, train accessibility, etc.). MAYBE email verification?
- **Location Manager/Directions View Controller Class:** This will essentially implement the location data gathered through the Core Location framework. Predominantly getting the User's current location and displaying it on a map. Should have the init() function, getCurrentLocation(), and viewDidLoad() methods to instantiate the CLLocationManager object. Will also include methods such as startUpdatingLocation(), setUpdateDuration(), stopUpdatingLocation(), and findPathTo(location). This class will serve as the main control class.
- **MapView Class:** This is the class that wil update and display the map of the surrounding area and will be in close relationship to the Location Manager class, CoreLocation framework, and MapKit framework for iOS. In fact, the Map class may just be implemented within the LocationManager class and it's methods.



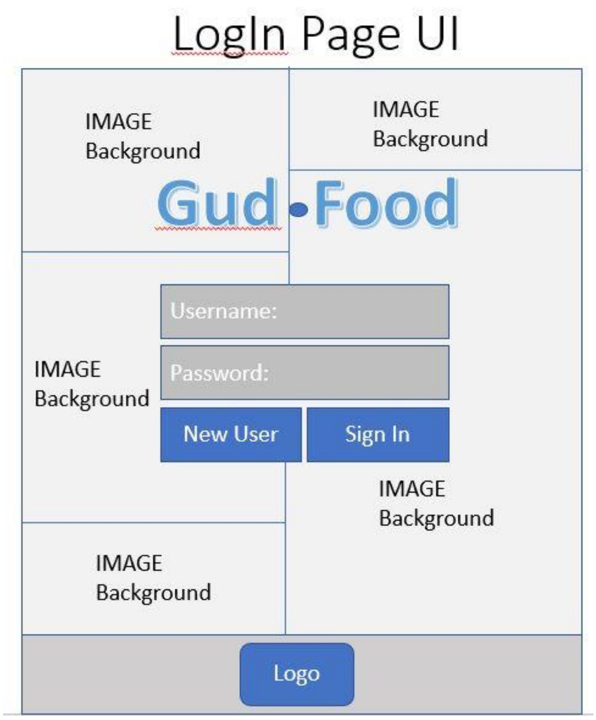
StateChart: Modeling the Overall Behavior of the System

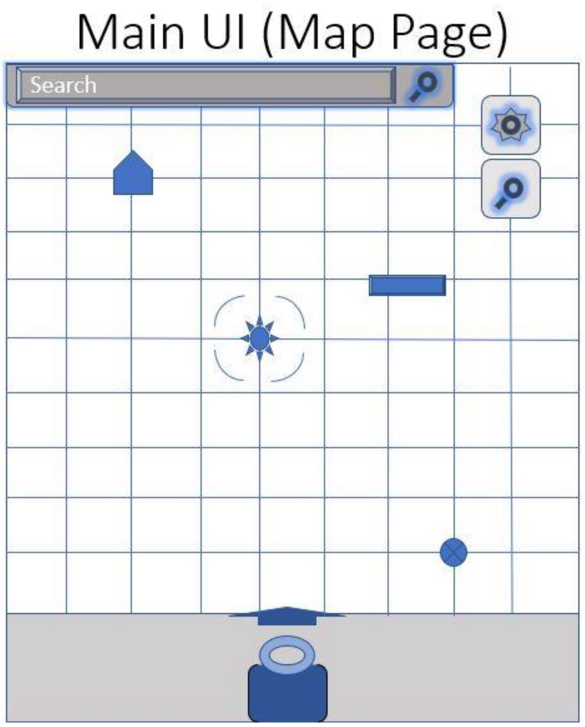

User
Select login
Validate User
System status = 'input ready'
displayMsg = 'enter user ID'
displayMsg = 'enter password'
User ID Validated
password validated
First time user?
Yes
Display User Profile Page
display: search preferences
display: accessibility options
Exit link pressed
select user profile
Display Map UI Page (Main page)
display: accessible locations/current routes
display: user actions (log out/ user profile)
display: search bar
select logout
cancel
successful creation of account
Display create account UI Page
displayMsg: 'create password'
displayMsg: 'create username'
displayMsg: 'email authentication'
Invalid parameters
select route
select 'clear routes'
update display map to default
Display location UI Screen
display: base class attributes
display: address
display: travel methods
display: Checks and X's if they go against specific constraints
display: distance
select accessible location

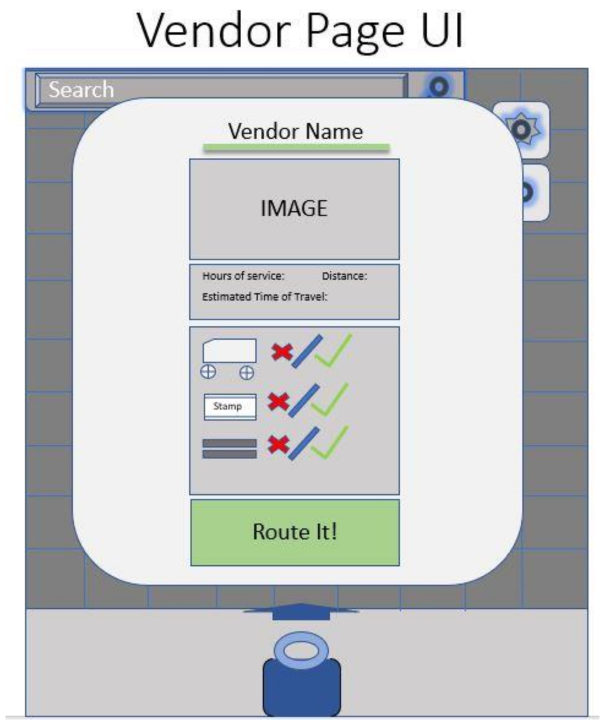
System Sequence Diagrams: Modeling the Overall Behavior of key User Cases

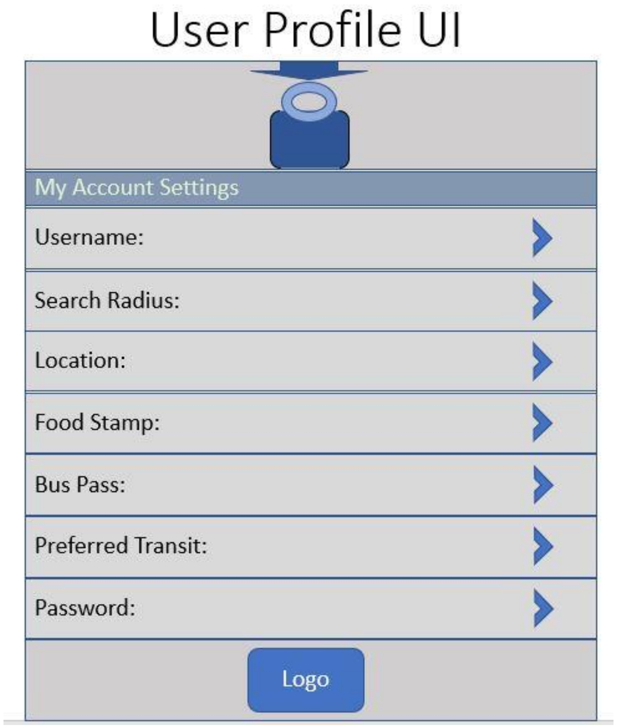


User Interface Design Pages (Mock-Up Design Concepts):









- **Strive for Consistency:** Each main UI screen will exhibit the same color scheme, and all subscreens will all just have have either a white background or faded background. The Gud-Food Logo will be shown at the bottom of screens that do not have the Edit Profile capability attached to it.

- **Enable Frequent Users to Use Shortcuts:** The app will allow users to automatically route to a destination without first going to the Vendor UI page, by pressing and holding the Vendor Icon on the map for 2 seconds at a time.
- **Offer informative feedback:** Upon changing user information/preferences, a check mark hologram will momentarily appear on the screen with text reading "success". When routes are generated, a confirmation tone will be played. When routes are successfully loaded, the user will have the option to view the direction in text.
- **Design dialogs to yield closure:** As mentioned, holographic check boxes or X's will be displayed upon successful or failed attempts to change user info/search preferences.
- **Offer simple error handling:** If the user enters invalid parameters for username/password with account creation, warning signs will be displayed next to the input fields with incorrect parameters.
- **Permit easy reversal of action:** All information related pages (in editing mode) will require a user to select "save changes" button, or "Cancel" to cancel their changes. When en-route to a destination, the "End route" button will restore defaults to the Map UI Page.
- **Support internal locus on control:** The only time that user will not directly be in control is when the location manager is getting/updating their current location, or when calculation routes to a destination.
- **Reduce short term memory load:** Account creation/user profile editing screens will "save" the latest changes made and display the most recent values in input boxes to the user, even if the user has not yet hit "save changes" as long as the input field did not have any input errors. So if the User messes up an input field, all other input fields are not erased when fixing.

Important Design Consideration:

- **Modularity:** One of the main goals of this project is to facilitate information hiding and reuse. To accomplish this, a good form and balance of modularity and encapsulation are recommended. The system functionality is split into modules that each take care of key features in such functionality. For example, each UI page view will be modularized into "exampleViewController.swift" where "example" is replaced by the corresponding UI page component. So some of the foreseen modules in my implementation would be a "PageViewController.swift" module to control which UI page is shown and at what times. An "AddressTable.swift" module to facilitate the address look-up table that will be presented to the user after they hit "search" from the search bar functionality. A "DirectionViewController.swift" which will facilitate the generation of routes to destination on the MapView UI Page. A "DirectionsTableView.swift" similar to the AddressTable.swift module to control and display the directions presented as step texts to the user. A "UserProfile.swift" module which will implement the UserProfile Class, a "Vendors.swift" module to implement the Vendor Class, and so on.
- **Encapsulation:** All classes contain data only pertaining to that class except for information that is gathered from other classes. For example, the user profile class contains only private attributes that are edited through the getters and setters for that

class, and the location attributes in the userProfile class is received from the getCurrentLocation() methods and locateMe() methods in the LocationManager class.

- **Appropriateness of Data structures used:** For most of the important information that is key to overall system functionality, the information is stored into Tuples of (String, String) or Tuples of (String, MKPlacemark) objects within the UserProfile class and LocationManager classes. This was chosen because this is information that I do not necessarily want be able to be changed by other processes, this is information that is final. Although the user's geolocation will change with each update duration, each instance of location data should not be changed. The user will only be displayed at one location at any instance in time on the MapView UI Page. Other than this, other data structures just include storing variables into appropriate data types, such as floats for geolocation latitude and longitude, and Strings for "phone number" attributes in the Vendor Class. For authentication and verification of user Login, I may need to implement an authentication database for this. A database that holds User information upon account creation, to verify whether or not a certain login parameter is correct, of if the user information trying to be created doesn't already exist.

Test Case Design:

- **Plans for Unit Testing:** Testing the getCurrentLocation() method in the CoreLocation framework to see if we can input a fake location (for simulation) and get that same location back in latitude & longitude is crucial. Without this method working properly, then the entire functionality utilizing location services is now unavailable, rendering the app almost useless. Testing the reverseGeocodeLocation() method in the CoreLocation framework to test if we can input a latitude and longitude coordinate and get back a human-readable address. This is also important because we need this is order to populate the AddressTable after the user enters a potential search location. Testing to see if locations can be displayed on the map using the MapKit framework's setRegion() method to be able to generate the general location the user finds themselves in is also essential. If this is not first established, then it will be impossible to further print routes onto this map. The Map will be blank without this functionality properly tested by itself.
- **Plans for Integration Testing:** Testing the getCurrentLocation() method in conjunction with the reverseGeocodeLocation() method is an idea. Seeing if the output from the getCurrentLocation() method could be used as an input for the reverseGeocodeLocation() is important because this is actually what needs to happen in order to automatically populate the "start location" field in the background when trying to establish routes to a specific destination because the user should not have to input a "start location". Another plan is to test the conversion of the CLPlacemarks returned from the CLGeocoder operations into an MKPlacemark. Then after testing the conversion between the MKPlacemarks into MKMapitems, to be able to display locations onto the Map screen. Finally testing to see if we can use these MKMapitems to run MKDirectionRequests to return MKRoutes from Apple.

- **Plan for System Testing:** Establishing a sequence of events to take place, built off of my System Sequence diagrams and attempting to replicate these sequences in one run-through. Essentially testing to see if the results of my Unit and Integration testing actually
- aided in the over performance of the system.

Functionality Tested	Inputs	Expected Output	Actual Output
Getting current location with Core Location	None	Simulation "fake location" coordinates Tuple(float, float)	
Translating location to human-readable addresses using reverse geocode location	Tuple(float, float) of coordinates Latitude and longitude	Array of placemarks with one value/element representing CLPlacemark – current address	
Process User Entries with Core Location (search bar functionality)	Address String	List of placemarks to select in AddressTable	
Calculating route with MapKit	Source + destination nodes	Array of MKRoutes objects	
Adding MKRoutes to MKMapView	Array of MKRoutes	Polyline of route printed on the map	
Printing MKRoute directions	Directions array of type Tuple(String, String, MKRoute)	Array of MKRoutes steps in DirectionsTable	
Changing user profile search preferences	New search parameters input fields	Updated screen displaying new search parameters	
Authenticate user Log-in information	Username + password	Permission denied if either field does not match authentication database information	
Input parameters only accept certain formats	Correct formats and incorrect formats entered.	Successful editing complete with correct format, input parameter indication of error if wrong format.	