VERSION 1.3

OCTOBER 20, 2016

CAMPUS CALORIES

SOFTWARE REQUIREMENTS SPECIFICATION

DYLAN INGRAM & AUSTIN PINKERTON

REVISION HISTORY

Date	Version	Description	Author(s)
09/15/2016	0.1	Initial draft with outline	AP & DI
09/19/2016	0.2	Rough draft with basic sections filled in	AP & DI
09/19/2016	1.0	Final draft with formatting included and all sections finished	AP & DI
10/10/2016	1.1	Began working on updated class diagrams and sequence diagrams and updating previous document	AP & DI
10/14/2016	1.2	Updated the class diagram and added descriptions to activity diagrams. Began working on system sequence diagrams	AP & DI
10/19/2016	1.3	Finished the diagrams and finalized remaining sections and formatting to match scope.	AP & DI

TABLE OF CONTENTS

1. PREFACE

This is the requirements documentation for the Android-based Campus Calories application. This document is intended for developers and users to guide them through the development during the course of the project.

2. INTRODUCTION

While the University of Alabama offers various websites that contains dining locations, menus, and nutritional information, there is no convenient way for a student to navigate through all of this information on a mobile device. Campus Calories is an Android application that at its core will allow users to navigate an interactive map of the University of Alabama's campus with dining locations marked as Points-of-Interest (POIs). These POIs can be selected by users to obtain various information (nutritional information, available menus, etc.) about the location. In addition to this basic functionality, we intend to include tools and frameworks that allow non-UA dining locations to enter their store information.

2.1 Product Overview

- Offers an interactive map for the users to navigate through to find pins that represent dining locations.
- Selecting a pin will bring up a small preview of the location with basic information about types of meals and hours
- Double clicking a location will bring up a more detailed store information screen with links to locally cached menus with nutritional information in addition to all the basic store info.
- The application will directly interact with a server side database. The database will set a flag on a per-store basis if their information was updated. The app will check for these flags on launch and will download the updated information for each store with a flag.
- Non-UA store owners will have the opportunity to add their store information, menus, and nutritional information on a separate website that pushes their information to our server.

2.2 Constraints

- The application's storage and processing will be limited to the individual's device.
- Our goal is to develop an app that requires minimal processing and storage requirements.

2.3 Assumptions and Dependencies

- The minimum Android OS we will be working on is Android 4.0 with an recommended OS of 5.0 and higher.

- Our GPS functionality requires a 3G/LTE connection, however, we would like to develop the application with an offline mode with all the store information cached. The stores can still be seen on a map, but the user's current location will not be available, and we cannot guarantee that the store information is updated to the most recent update without a network connection.

2.4 Maintainability

- Campus Calories is being developed in conjunction Dr. Reed, an Associate Professor, from the University of Alabama.
- After completion, we will transfer the ownership to her for her to develop at her discretion in the future.
- With that in mind, our system must be developed in a way to facilitate and foster additional development and feature expansion.

3. GLOSSARY

Android: A software stack for mobile devices that includes an Operating System and key applications that server as the intermediary between the hardware and the user.

Application: Commonly abbreviated as "app," an application refers to a single purpose piece of software. In our use, it describes our mobile device application, Campus Calories.

Framework: An abstraction in which the software package provides generic functionality that can be changed by additional code which provides additional functionality. Frameworks add to the overall modularity of a system.

Application Programming Interface (API): A particular set of rules and specifications that allow for the interaction between software components in a program.

Server: A computer or computer program that manages access to a centralized resource or service in a network.

3G/LTE: A cellular data connection type that will be used to connect to a GPS signal and our server-side database.

4. USER REQUIREMENTS DEFINITION

4.1 Functional Requirements

- A user shall be able to navigate a campus map to find dining location.
- A user shall be able to select and view a dining location with updated menus and nutritional information.
- The system shall be able to connect to a server to check if there is newer information about the dining locations to download.
- The system shall include an option for restaurants to create their own menus and add the nutritional information for them.

4.2 Non-Functional Requirements

- The system must be able to operate without a network connection or a GPS signal outside of the initial application opening.
- The system must be able to run on low-end devices.
- The system must not be a significant drain on the battery life of devices.
- The system must not require a large amount of data transfer after the initial download
- The system must be able to find the location the user specifies as well as any relevant information about the location.

5. SYSTEM ARCHITECTURE

5.1 Main Application

- An interactive map (i.e. able to be moved, zoomed in/out, etc.) with dining locations marked as pins. The main screen should also contain a search bar at the top with options to select dietary restrictions (e.g. Vegetarian, Vegan, etc.).
- The dining locations should also be marked on the map. If the user selects a point, a popup with basic info will appear above the location. If the user decides he/she wants to learn more about this location, they can tap the popup box which will bring them to a more screen with more details on the location.
- The app will include the background processes to connect to our server to update information on locations and location information.

5.2 Server-side Database

- Scrapes the UADish website for updated information and stores if the data is newer than last scrape. The database will also accept and store information entered on the website that allows a non-UA dining location to add in their information. The database will set a flag if the information is updated for a specific location. This will prompt the app to download the updated information when opened.

- We are currently still exploring options and funding for our server and database software

5.3 Website

- Offers the options for a non-UA dining location to enter store information. This information includes but is not limited to store name, store location, hours of operation, menu types (breakfast, lunch, dinner), menu items, nutritional information of menu items, and dietary restriction flags.
- We will also offer the option to view/edit the store information by the non-UA dining location manager/owner. This will require the use of authentication and login/password storage that we are still investigating.
- This information will be pushed and stored on the database located on the server.

6. SYSTEM REQUIREMENTS SPECIFICATION

6.1 Data Class

- The Data class will offer methods of locally storing and querying information stored on the server.
- The information that is cached locally includes all the basic store information such as hours of operation, telephone, and store location as well as all the menu/nutritional options
- The menu is separated into types, breakfast, lunch, and dinner. From there, sections are added to the menu that include appetizer, entrée, dessert, etc. Finally, the food items themselves are listed with relevant nutritional information. Flags are also included for food types indicating dietary options such as vegan or vegetarian.

6.2 Map Class

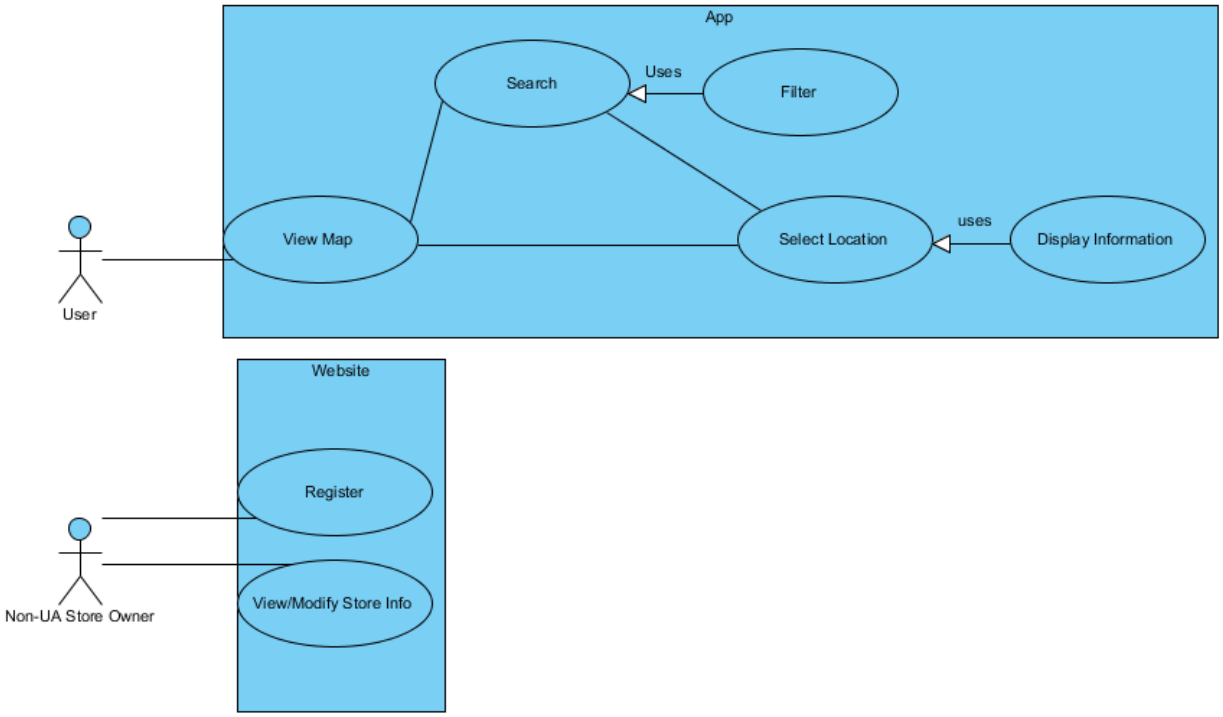
- The Map class contains the map info for the University of Alabama's campus. This map will be the basis for our pins located on dining location.
- Our ideal API to use would be Google Map's API due to its ease of development and supported features.

6.3 Hardware Class

- These Hardware classes will implement the connections between the Main App and the server as well as offer a framework to add support for future development.
    - Communication Class: This class will be used to interface between the main app and our server. This class will handle all the data querying and transfer from server storage to local storage.
    - GPS Class: This class will be used to track the user's location to give a relative distance between the user's location and the dining location he/she chooses.

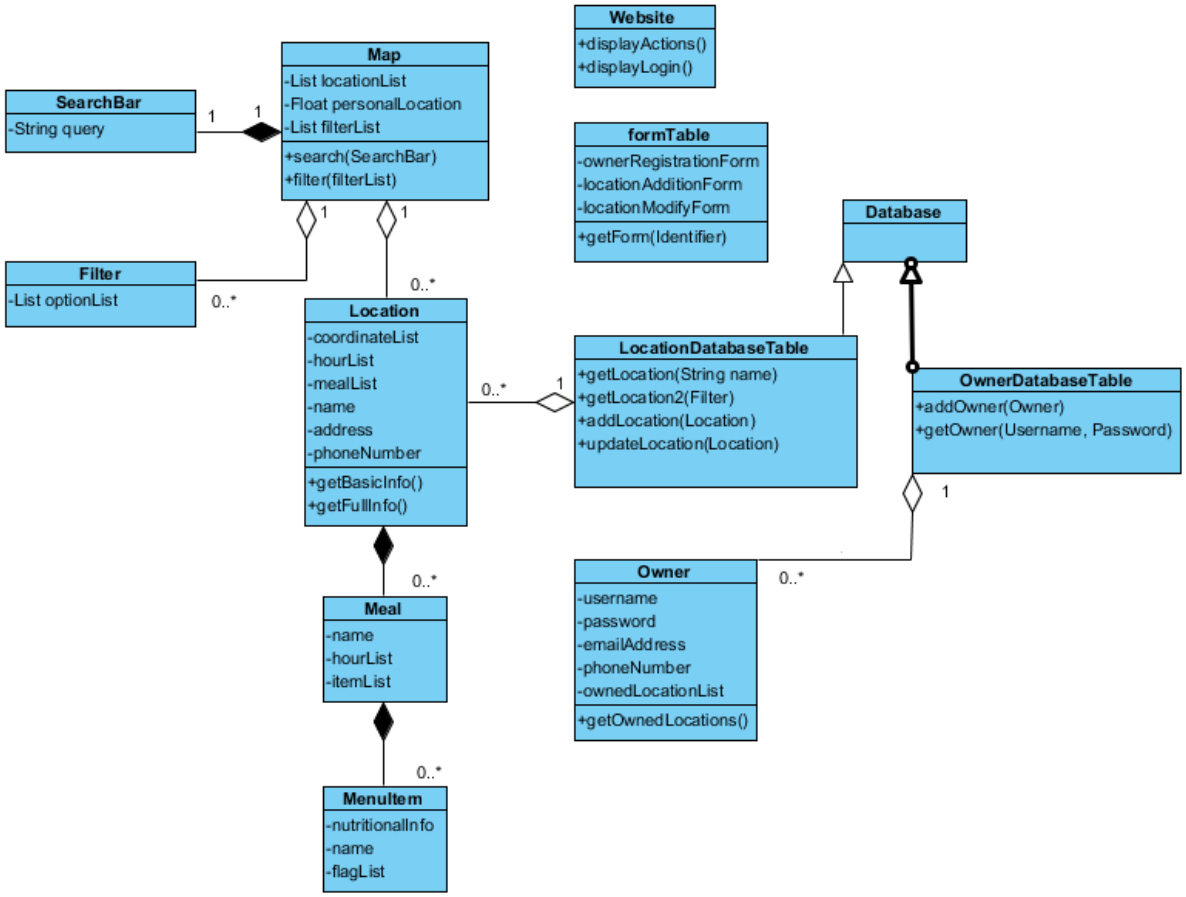
7. SYSTEM MODELS

7.1 Use Case Diagram



Use Case Diagram for both the mobile app user and non-UA store owner. A mobile user will get the map-based UI persistently and can request to search for a specific location with a string or filter locations based on dietary restrictions. The store owner uses the website UI and can register for an account, view/modify, and add locations they own to our databases.

7.2 Detailed Class Diagram



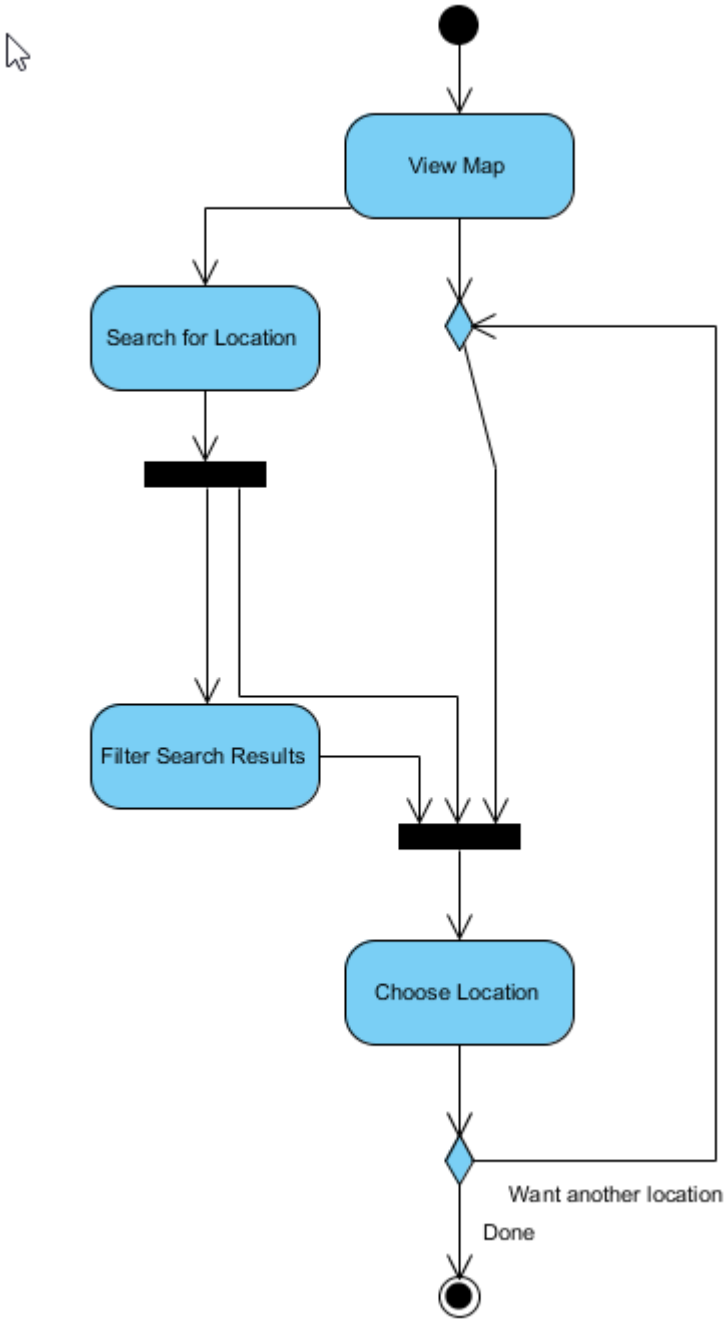
Our updated main class diagram with classes and methods detailed. Map and Website will be the main UI elements for the mobile app user and store owner, respectively.

Our map-based application will have two options for the user who wants to specify information. They can filter all locations based on various dietary restrictions or they can search by name for locations containing that search string.

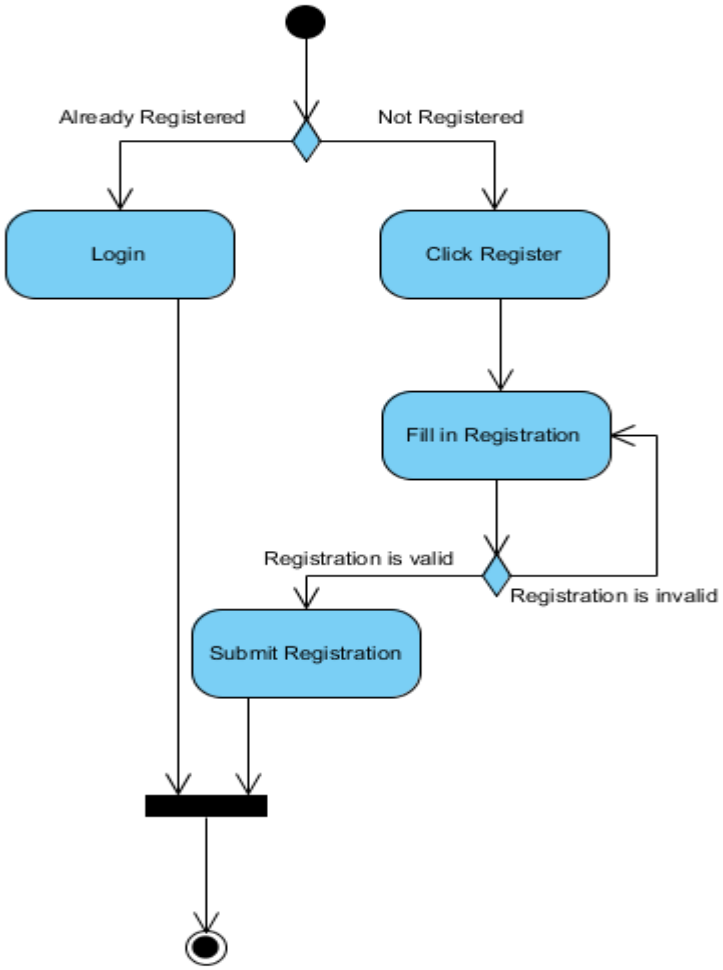
The website will allow the owner to register for an account and view/add/modify all locations they own.

Both the location database and owner database will inherit from the database class. Meals and MenuItem will inherit from the Location class.

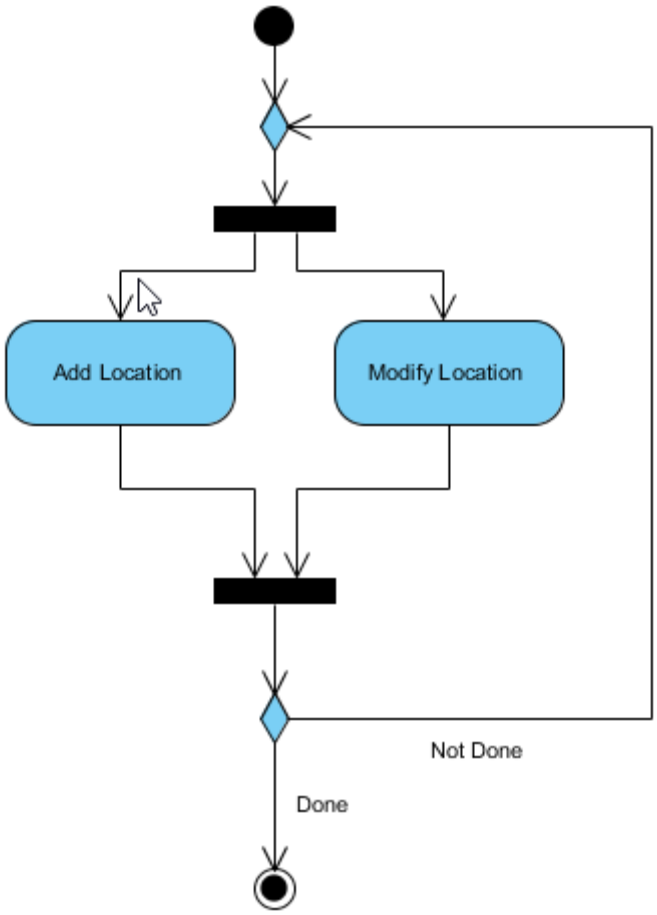
7.3 Activity Diagrams



The Activity Diagram for a user finding a store location and information. The user can either search for a location by name and retrieve a list of stores containing that string or filter all locations by various dietary restrictions.

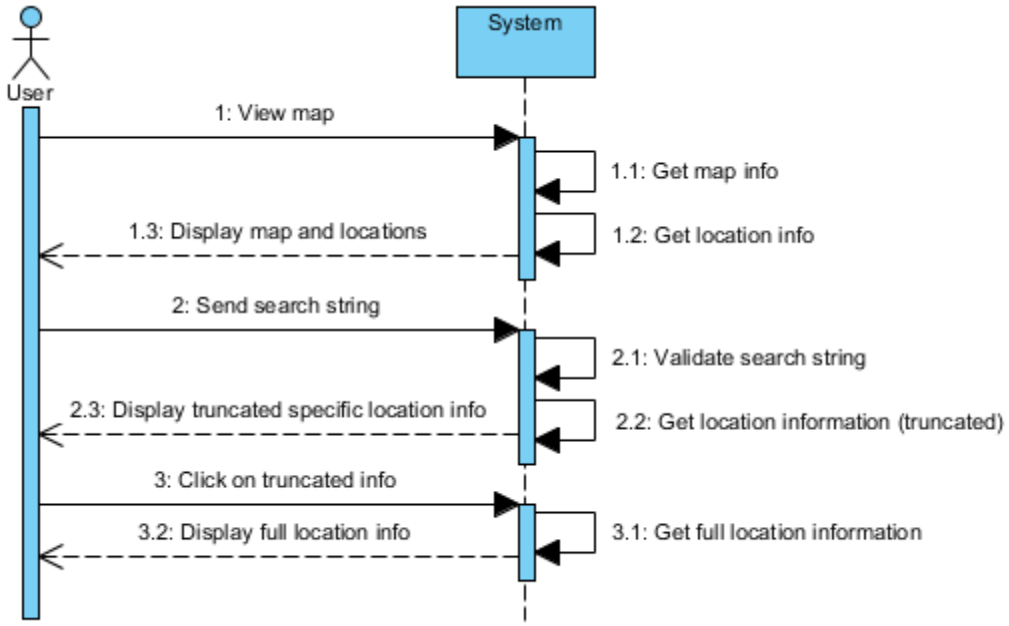


The Activity Diagram for a non-UA store owner to log or register for our system. An owner can either contact us or we can contact interested owners to receive a validation code that they will use when registering. This will create an added layer of security to prevent anyone without a code from registering. After successfully registering with a form, the owner will be created and added to our Owner Database.

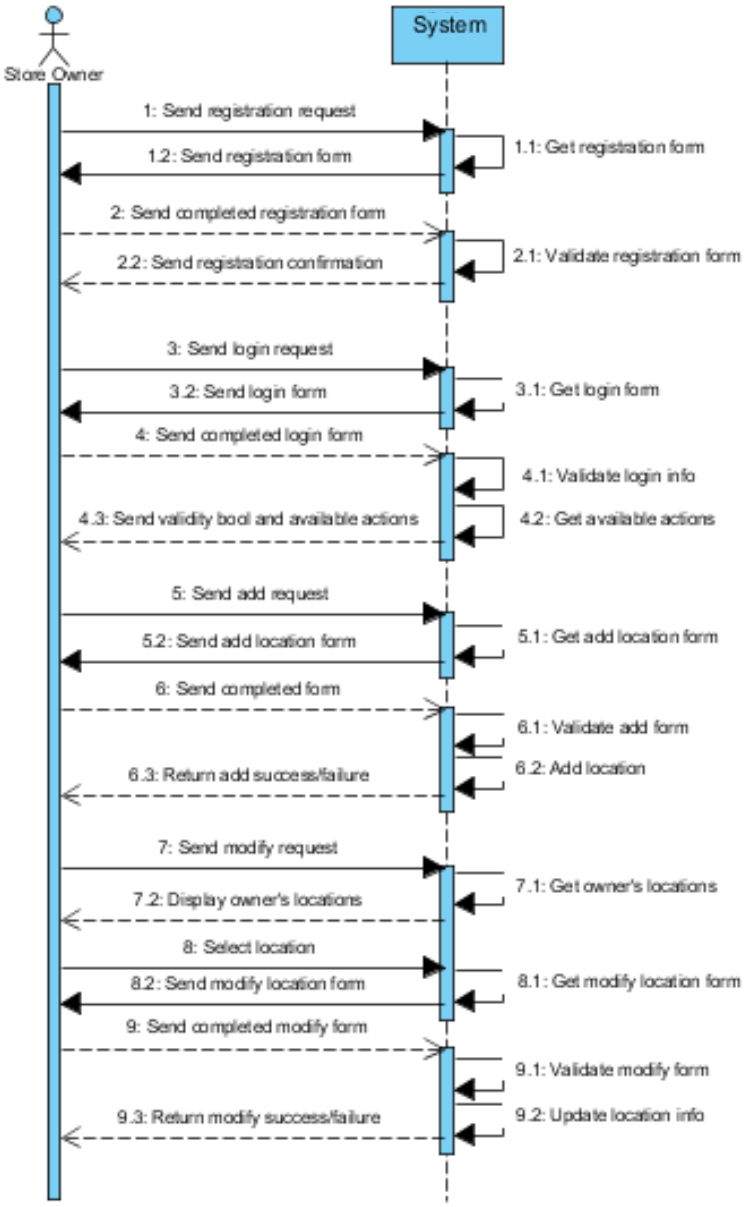


The Activity Diagram for a non-UA store owner to add or to modify their store's information. An owner can add or view/modify their current locations. Once a location is created, it is pushed to our Locations database as well as the current Owner's class.

7.4 Sequence Diagrams

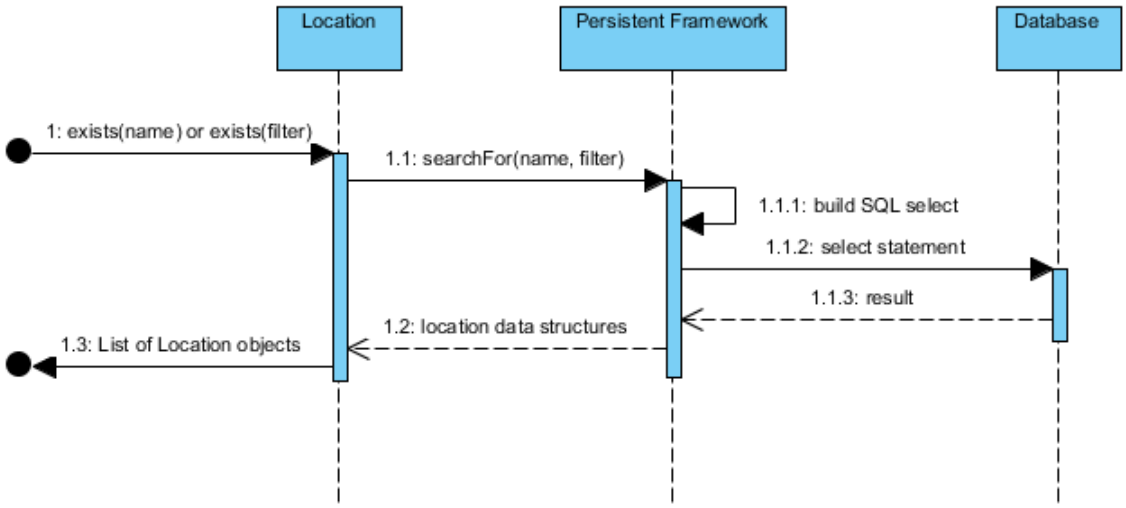


The System-level Sequence diagram for a mobile application user wishing to search for a location. In this diagram, the system represents the map-based UI of the application and the location database. Upon application opening, a view map request is sent and the relevant map and location information is found and displayed. The user can search for a location using a string, the string is validated, the string is passed to our getLocation method, and a list of locations containing the string is shown to the user. This is similar to our filter feature, but instead of a string, a filter object is sent to the getLocation() method.

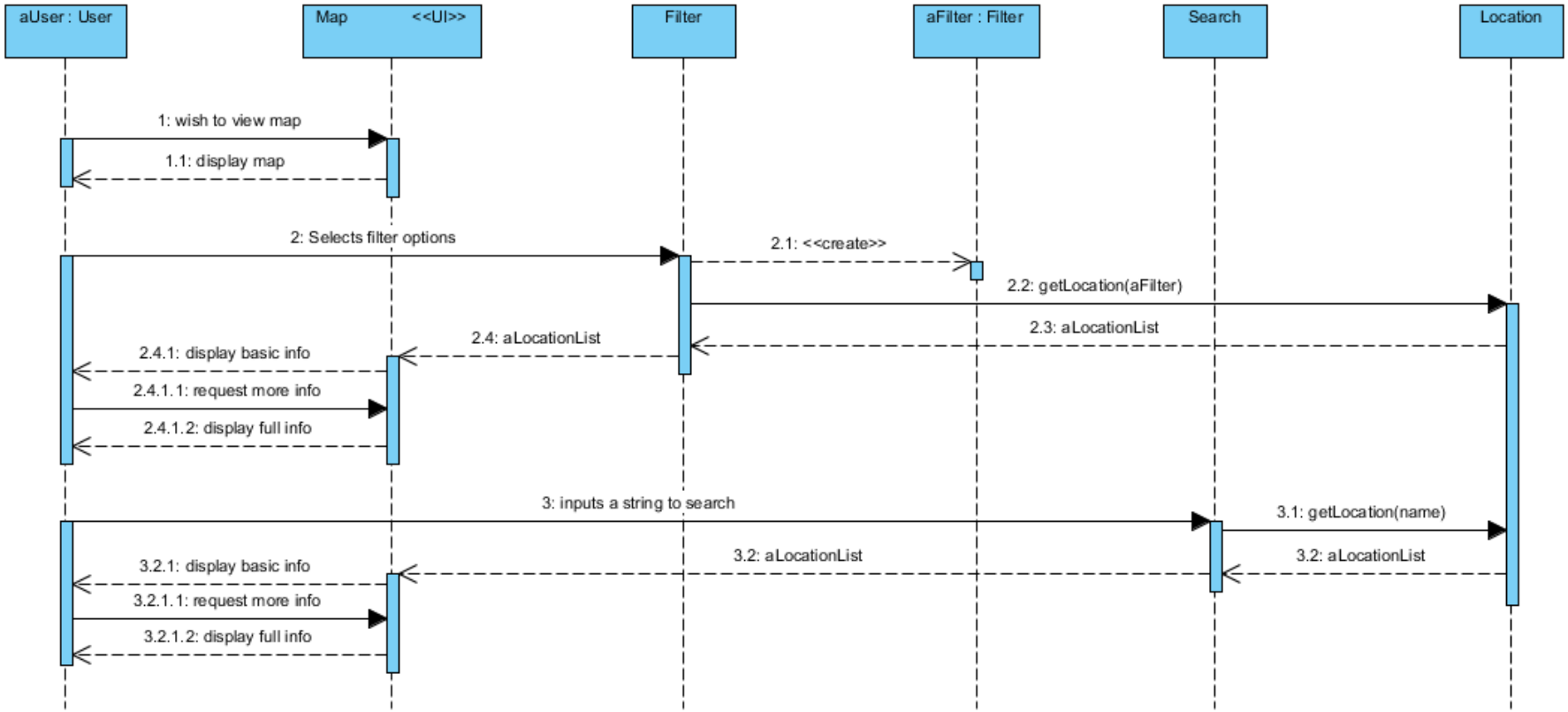


The System-level Sequence diagram for a store owner wishing to perform various actions through the website. For this diagram, the System represents our website, the Owner database, and Locations database.

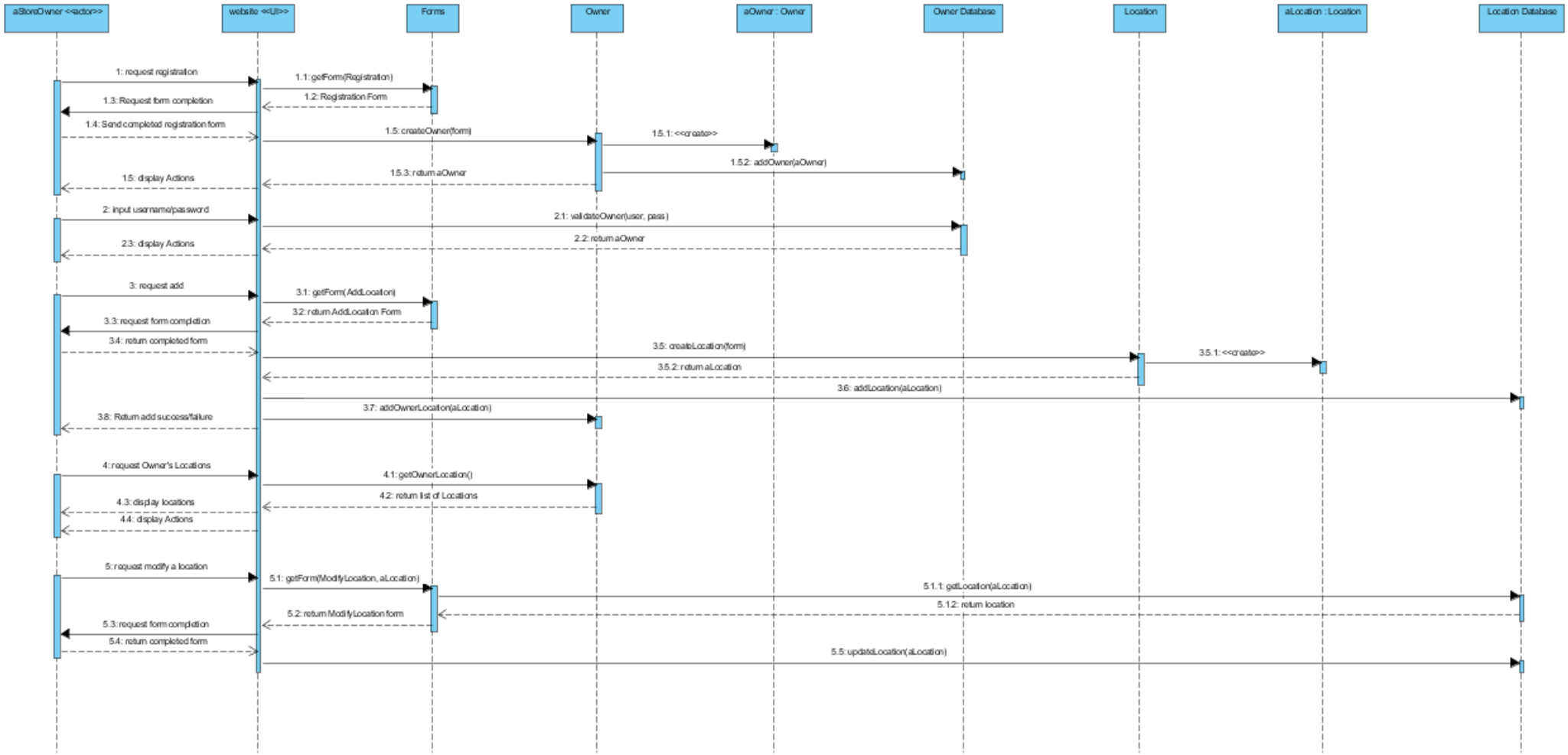
1. This represents a request to register sent from the owner
2. Returning filled in form to the system
3. A login request
4. A login confirmation and displaying available actions
5. Adding a location request
6. Returning a filled in form for a location, adding the location to the DB, and a confirmation
7. A modification request that first displays the owner's owned locations
8. The actual modification request form sent to the owner
9. The updating of the location in the DB and the confirmation return.



A Service-level sequence diagram that represents the getLocation() method. This method is crucial to both the application and website UI as both require specific location requests. This method validates a search, creates a statement for our DB, and returns a list of the results. It can take in a string that represents a name of a location or a filter object that uses a list of dietary options to filer with.



A Course of Action Sequence diagram for a mobile application user. This takes the basis from the System-level User Sequence diagram and expands upon it with methods and classes.



A Course of Action Sequence diagram for a store owner. This takes the basis from the System-level Owner Sequence diagram and expands upon it with methods and classes.