University of Redlands

Managing Marine Mammal Observations Using a Volunteered Geographic Information Approach

A Major Individual Project submitted in partial satisfaction of the requirements for the degree of Master of Science in Geographic Information Systems

by

Melodi C. King

Douglas M. Flewelling, Ph.D., Committee Chair

Lei Lani Stelle, Ph.D.

December 2012

Managing Marine Mammal Observations Using a Volunteered Geographic Information Approach

The report of Melodi King is approved.

Lei Lani Stelle, Ph.D.

Douglas M. Flewelling, Ph.D., Committee Chair

December 2012

Acknowledgements

First and foremost, I would like to express my sincere gratitude and thanks to my advisor and committee chair, Doug Flewelling. Doug consistently gave me creative freedom during development and acted as a sounding board and mentor without being overbearing with the project.

I would also like to thank my client, Dr. Lei Lani Stelle. I could not have asked for a better client relationship for my major individual project. Lei Lani was enthusiastic about the project and always made herself available when I had questions. Dr. Stelle and her husband, Shane Keena, always made me feel very welcome and comfortable when we were out on the boat. A special thanks to Shane for the wonderful photos donated to the project and the adventurous whale watching trip.

Of course, the whale watching trips would not have possible without the help from Captain Larry of Davies Locker Whale Watch. Captain Larry always made his boat available for me if I needed to test the mobile application.

Additional thanks to the entire MS GIS department faculty and staff. A big hug and thanks to Debbie Riley for always making sure that I had a cup of green tea. Doug Flewelling, Mark Kumler, Ruijin Ma, and Fang Ren provided me with an unforgettable learning experience that I will always treasure, and showed unconditional patience with me and my plethora of questions. A special thanks to Mark for letting me create my own learning experiences, the many brainstorming sessions of better communication with graphics, and for not hating me after I requested a different advisor.

I'd like to thank my family and friends for their unconditional love and support while I've taken the time to find and pursue the career path that was right for me. A special thanks to my brother, Matt King, for helping me with graphics. I look forward to future collaborations with you.

One last thank you to Cohort 21. We've been through a lot this year, and I'm so thankful for each and every experience I've had with you.

Data is still King

Abstract

Managing Marine Mammal Observations Using a Volunteered Geographic Information Approach

by

Melodi King

Traditional methods of gathering the data needed to understand human impact on marine mammals requires extensive time and resources. To reduce the burden associated with collecting and managing marine mammal observations, a geographic information system (GIS) solution was developed using a volunteered geographic information (VGI) approach. Web and mobile applications were built for the general public to submit marine mammal observations and visualize the results. The web application also includes querying and authorized download of data. Both applications consume web services published from an ArcSDE geodatabase using ArcGIS Server 10.0.

Table of Contents

Table of Figures

List of Tables

List of Acronyms and Definitions

API	Application programming interface
CINMS	Channel Islands National Marine Sanctuary
CWA	Coastal Web Atlas
EBM	Ecosystem-based management
GIS	Geographic information system
GPS	Global Positioning System
GUI	Graphical user interface
IDE	Integrated development environment
IIS	Internet Information Services
NOAA	National Oceanic and Atmospheric Administration
OBIS-SEAMAP	Ocean Biogeographic Information System-Spatial Ecological Analysis of Megavertebrate Populations
PPGIS	public participation GIS
SDE	Spatial Database Engine
SDK	Software development kit
SOD	Sudden oak death
URL	Universal Resource Locator
VGI	Volunteered geographic information

Chapter 1 – Introduction

The oceans of the world play an indirect but fundamental role in life; they are used for activities such as shipping, procuring food, recreation, and travel. Their uses have also expanded to include renewable energy and large scale aquaculture. However, it was not until recently that scientists began establishing a scientific baseline for evaluating the health of the marine ecosystem (Ruckelshaus, Klinger, Knowlton, et al., 2008). This baseline is particularly important to recent efforts in evaluating principles for marine spatial planning for effectively managing marine resources (Foley, Halpern, & Micheli, 2010).

In order to understand behavior of, and human impact on, marine mammals, extensive manpower for collecting and processing data is required, due to the complex interactions between marine mammals, humans, and oceanic processes. The burden associated with this type of research can be reduced using a volunteered geographic information (VGI) approach with a geographic information system (GIS). Incorporating familiar user interfaces, such as web and mobile applications, allows researchers to spend more time performing analyses while simultaneously encouraging awareness and environmental stewardship in users.

This chapter was designed to introduce the reader to the project. Section 1.1 introduces the client. The second section, 1.2, defines the problem addressed. The proposed solution, including the goals and objectives, scope, and methods are discussed in Section 1.3. The fourth section, 1.4, outlines the target audience for this report. Finally, Section 1.5 sets the expectations for the remainder of the document.

1.1 Client

Dr. Lei Lani Stelle is a biology professor at the University of Redlands whose research pertains to the human impacts on marine mammals and their habitat use. Specific components of her research include evaluating marine mammal species' associations, determining swim paths and behaviors, assessing vessel-induced injuries, and understanding energy expenditures of marine mammals during a migration path affected by human interactions.

Over the course of the project, Dr. Stelle was responsible for describing the types of data that were collected and verifying that the system design met her needs. Additionally, she was responsible for approving the user interface design of the web and mobile applications during the testing and discussion tasks of the project's life cycle. Finally, the client acted as a domain knowledge expert when questions arose during design and development of the solution.

1.2 Problem Statement

The challenge that the client faced was how to generate a high volume of quality data for her long-term study on marine mammal migratory behavior and human impact. In order to reduce the cost and effort required in the data collection process, Stelle decided to

explore a VGI approach to collecting this data, which incorporates the collection of data by both researchers and members of the general public (Figure 1-1).

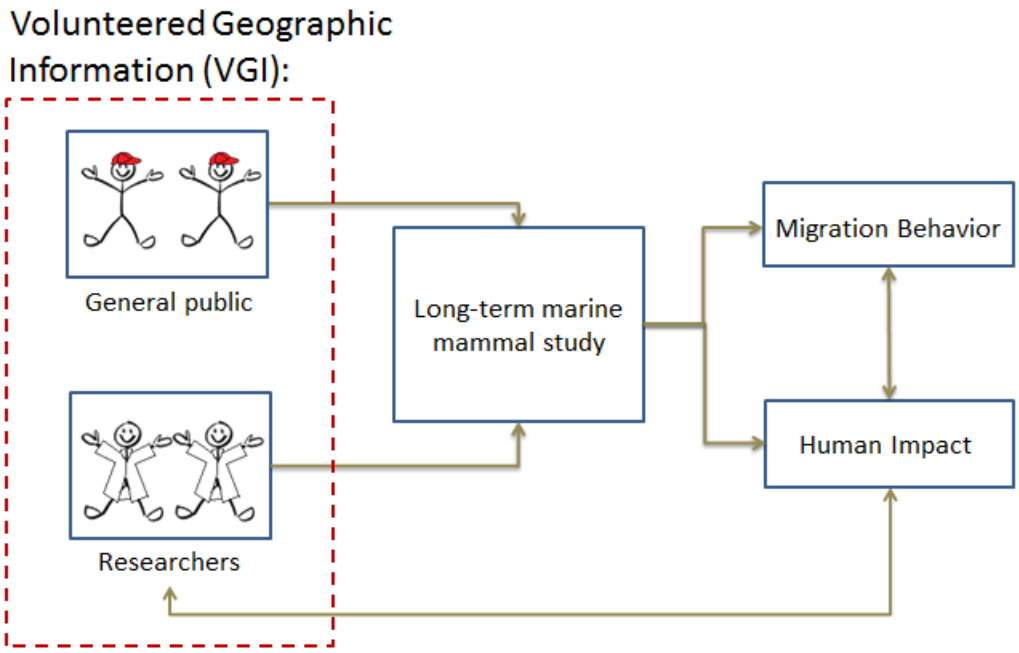


Figure 1-1: VGI-approach for collecting data for a long term study of marine animals

Currently Stelle works with volunteer citizen scientists through Earthwatch, an organization that facilitates participation by members of the public in the scientific process. Additionally, Stelle collects data with undergraduate students performing research for thesis projects. She believes that by including the general public in the data collection process she will further reduce difficulties in maintaining long-term studies while simultaneously encouraging awareness and environmental stewardship.

Although including the general public in the data collection process would increase the amount of data collected, it would also introduce questions of data quality. The client has very few protocols in place for ensuring high data quality from each of the sources. For example, currently, data are hand-written, leaving room for error during the process of transferring them from paper to electronic source. Without continuity and consistency in the data collection process, an unnecessary amount of time has to be spent preparing and organizing data over the course of a long-term study.

With the approach developed in this project, Stelle would be able to focus efforts on understanding the relationships between migrating mammals and humans, in addition to improving the learning experience for the students and volunteers she works with. Specifically, because of the nature of GIS databases, data collected by volunteers will have continuity and consistency. Having a centralized database that can be queried would allow Stelle to collaborate with other researchers. Additionally, having the data stored in spatial tables allows her to ask advanced spatial questions.

1.3 Proposed Solution

After careful review of previous work and consideration of the client's requests, a solution was proposed. The following section outlines the proposed solution and its appropriateness for the client. It includes a discussion of project goals and objectives, scope, and methodologies used to develop the solution.

1.3.1 Goals and Objectives

The two problems addressed in this project provided the client with geographic workflows for managing and sharing her data. The first problem was how to generate a volume of quality data for a study on marine mammal migratory behavior and human impact. This problem was solved using a VGI-based strategy which incorporated the general public in the data collection process to increase the amount of data being collected. More specifically, mobile and web application prototypes were developed that can be used by volunteers, researchers, and members of the general public to submit marine mammal observation data. Web and mobile technology were chosen because they were already demonstrated to be successful in a VGI-based study (Connors, Lei, & Kelly, 2011). This solution decreased the amount of time researchers and volunteers spent in the field and increased the amount of data collected. It also provided the client with the data necessary to perform her research.

The second problem addressed in this project was the data management methods. The client stored data in Microsoft Excel, Access, in species-specific programs, and in programs developed for specific projects. Without proper management of the data, a large amount of time was spent on organization efforts and preparing the data for analysis.

This problem was solved with the development of a centralized geodatabase used to manage data submitted from student and professional researchers, volunteers, and the general public. The geodatabase was used to house marine mammal observations. The outcomes of the solution were time savings to the client and possible identification of marine mammal observation data standards. The development of a geodatabase allowed Stelle to spend less time preparing data for analysis, more time improving the learning experience for her students and volunteers, and collaborating with her colleagues in defining a baseline for evaluating the health of the marine ecosystem.

1.3.2 Scope

While the purpose of this project was to demonstrate the feasibility of using a VGI approach to managing marine mammal observation, only a very basic, but extendable, solution was developed. The scope included the development of web and mobile applications that allows the user to submit and visualize their observations. The solution also included a geodatabase for storing the submitted observations.

The Spatial Database Engine (SDE) geodatabase, a type of relational database management system, was developed for use with the ArcGIS Server 10.0. The geodatabase was designed to hold observation data and corresponding evidence and to tie observations to the user who submitted them, through the use of usernames. However, the proof of concept was designed so that users had the same level of access. There were no

database permission differences between the experts (researchers, students) and general public users.

Due to the time constraints of the project, the scope of the database development was limited and client expectations were clearly defined. The geodatabase's schema was developed and its capability was demonstrated with sample data. However, cataloging of the current data was the client's responsibility. Additionally, the client was responsible for describing the types of data that were collected and verifying that the geodatabase's schema met their needs.

The web application was developed to allow users to visualize, query, and download data. Specifically, it was designed to allow users to visualize and query the entire database by date and species type. Upon logging in, the application allowed users to query by event type (observation, track update) and choose to visualize all of the data in the database or only their data. The web application was also developed to allow users to download data upon log in to the system. The users' identity was not verified when logging into the system with this initial concept. Finally, forms were developed to allow users to enter data that they collect in the field.

The mobile application's functionality was limited to data submission and visualization. While there were several platforms for which the mobile application could be developed, it was only developed for one. The mobile application was designed to store collected data locally, in addition to syncing the data with the geodatabase on the server. The client was asked to help design the appearance of the mobile application.

The solution was designed for use between Newport Beach and Long Beach (Figure 1-2) in southern California. The project was assigned a spatial scope for multiple reasons. Keeping the region focused resulted in a small list of species native to the region that the user would need to choose from. This small area also had reliable connectivity to the mobile network, which supported an environment to test the data syncing.

Figure 1-2: Map of the project's study area

1.3.3 Methods

The project was split into three different packages: the geodatabase, the web application, and the mobile application. Each package went through a staged developed life cycle (Figure 1-3).

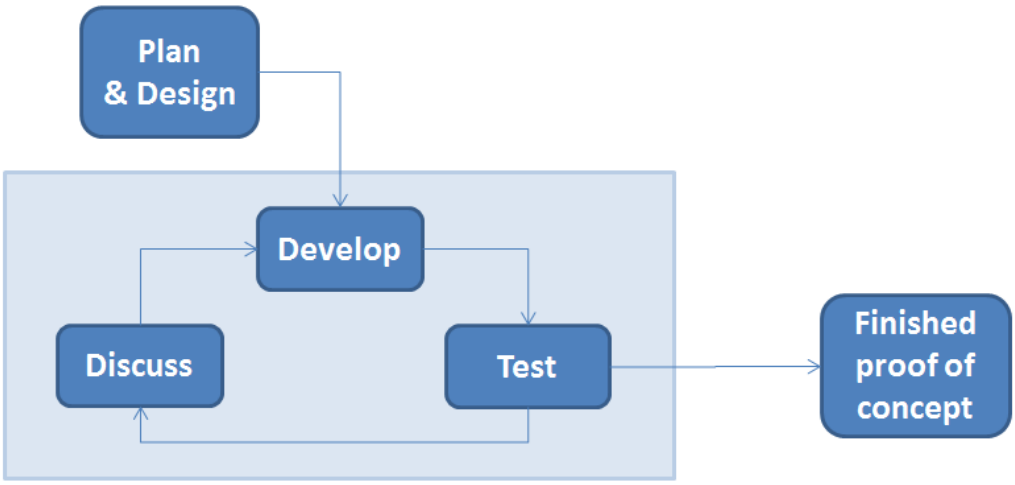


Figure 1-3: Project methods life cycle diagram

During the plan and design phase for the geodatabase, requirements were classified into functional and nonfunctional requirements. Conceptual model and logical models were then designed. In the development phase an SDE geodatabase was created. Additionally, map and feature services were published using ArcGIS Server 10.0 for consumption by the web and mobile applications. The logical model went through test and discussion phases before the finished proof of concept was reached. Each time there was a change in the database's schema, changes were made to a small set of test data.

During the plan and design phase for the web and mobile applications, the appropriate technology for development was decided. The web application was developed using Esri's ArcGIS Application Programming Interface (API) for Javascript and the mobile application was developed using Esri's Android Software Development Kit (SDK). During the develop, test, and discussion phases of the web and mobile applications, each of the functional requirements was built and pieced together to get the final proof of concepts. The develop phase for both of the applications included researching the APIs' classes and corresponding methods.

1.4 Audience

The intended audience for this report includes individuals who have an introductory knowledge of GIS, have a basic understanding of programming concepts; are interested in data collection using a VGI approach, or are interested in data collection techniques for marine mammal research. No specific knowledge of ArcGIS or programming is assumed.

1.5 Overview of the Rest of this Report

The remainder of the report describes how the project components were implemented. Chapter 2 provides a literature review of relevant topics. Chapter 3 describes the system design of the project. This is followed by a discussion of the database model in Chapter 4. Chapter 5 refers to the implementation of the web and mobile applications. Chapter 6 describes the lessons were learned during the software development process and the analysis that can be done with the data collected. The report closes with Chapter 7, a conclusion and discussion of future work.

Chapter 2 – Background and Literature Review

A literature review was performed during the project planning phase, during which four relevant topics were chosen. Because of the Volunteered geographic information (VGI) nature of the project, Section 2.1 is dedicated to understanding its use in web and mobile GIS as a means of collecting data. Section 2.2 presents the use of GIS in marine research. Section 2.3 discusses the differences between web and mobile GIS technologies and their appropriate applications. Designing user interfaces that can reach a broad audience is important in VGI. Because of this, user interface design is discussed in Section 2.4. The chapter is concluded with a summary in Section 2.5.

2.1 Volunteered Geographic Information and Science

Volunteered geographic information is closely tied to citizen science, which is the involvement of interested members of the public in parts of a scientific project such as data collection and analysis. It has been utilized in applications such as recording bird observations (The Cornell Lab of Ornithology, 2011) and online game playing in understanding protein folding (UW Center for Game Science, 2011). Allowing citizens to participate in the scientific inquiry process may bring about awareness, empowerment, and stewardship. Additionally, the inclusion of citizen scientists may help reduce the gaps that have historically divided the public, researchers, and policymakers in environmental management efforts (Connors, Lei, & Kelly, 2011).

A similar concept is public participation GIS (PPGIS). PPGIS is strongly focused on engaging citizens in the sustainability of their communities. "It is an interdisciplinary research, community development and environmental stewardship tool grounded in value and ethical frameworks that promote social justice, ecological sustainability, improvement of quality of life, redistributive justice and nurturing of civil society," (Aberley & Sieber, 2002).

Goodchild (2007) coined the term volunteered geographic information to describe geographic data provided voluntarily by individuals. The development of Web 2.0, Global Positioning System (GPS), and the rapid assimilation of mobile technology made VGI practical. Web 2.0 resulted from the development of protocols that made the communication between user and server a two-way conversation. This enabled users to create and edit information stored on the servers through the browser interface. In the 1980s the GPS was developed. Originally created for military purposes, the GPS made its way into the hands of the public around 1990. GPS allows for quick and easy direct measurement of locations on Earth and has been used in a wide variety of applications (Goodchild, 2007).

The use of mobile and web applications that utilize a VGI-based strategy in the collection of data in long-term environmental studies is a relatively new field. There is a working prototype of this data collection method called OakMapper, which was developed at University of California at Berkeley. OakMapper is a mobile (iPhone) and web-based effort to encourage the public in monitoring the sudden oak death (SOD) of oak trees in California caused by the ramorum leaf blight *(p. ramorum)* virus (Geospatial Innovation Facility, 2012).

Advancements in web and mobile GIS technologies that utilize a VGI strategy for scientific data collection are limited. Glennon (2011) created the Geyser Notebook application for Android. The application allows users to view information about the Yellowstone geysers and report eruption observations. User accounts are created, but there appears to be no differentiation between a researcher and a member of the general public. Within Geyser Notebook, a timeline shows observations from all users, and "my reports" shows only the user's observations. In addition to a mobile application, a web application has been developed that allows users to view the data stream (Glennon, personal communication, 2012). However, there is no evidence that this application is utilized by researchers in understanding geyser activity in Yellowstone National Park.

2.2 GIS & Marine Research

The complex nature of marine studies poses a unique challenge for researchers and decision makers. "Traditional management strategies, which focus on individual sectors of coastal ecosystems, such as managing single species habitat, or areas, have failed to address these intricate relationships between humans and coastal ecosystems," (Bauer, 2012). More integrated and comprehensive management strategies, such as Ecosystem-based Management (EBM), are being developed to address this problem (Jones & Ganey, 2009). Geographic information systems are being used in data collection and management in addition to analysis relevant to marine research.

2.2.1 GIS and Marine Research

Several efforts have been made to make data collected more widely available to researchers and decision makers. The Ocean Biogeographic Information System Spatial Ecological Analysis of Megavertebrate Populations (OBIS-SEAMAP) is a spatially referenced online database, aggregating marine mammal, seabird and sea turtle observation data from across the globe and have aggregated data since 2002. The OBIS-SEAMAP uses of geospatial web feature services. Ideally, "this makes data easy to use by modelers in a scientific workflow," (Best, et al., 2006).

The OBIS-SEAMAP is just one example of geospatial web services being used by researchers and decision makers in the field. Coastal Web Atlases (CWA) are increasingly popular web-based tool. The California Coastal Atlas was initiated in 1993. Its primary goals are to create a platform for sharing high quality coastal data, provide a medium for information sharing between scientists and public policy makers (University of Washington Sea Grant Institude, 2011). Additionally, the National Oceanic and Atmospheric Administration (NOAA) Coast Watch has developed a browser for downloading contour, grid, and vector datasets. The data available include: currents, chlorophyll, sea surface temperature, and wind stress to name a few (NOAA, 2011).

While sites like OBIS-SEAMAP, California Coastal Web Atlas, and NOAA provide a wide range of uses, there are also organizations that have made a more focused effort of collecting and sharing data. Oregon State University has made chlorophyll and temperature data available through an Ocean Productivity website (O'Malley, 2010). Although the data isn't available for download and there is no mapping component, the Channel Islands National Marine Sanctuary has a website in which users can submit marine mammal observations (National Ocean Service, 2011).

Essential to data collection and sharing is the data storage. The use of geospatial web services provides a method for accessing and downloading data stored on servers, but they do not address the need for best practices for storing data collected in the field and data downloaded from web services. Geodatabases provide components, such as subtypes and domains, to better manage spatial and non-spatial data.

In addition to the geodatabase are data models. Data models are schemas for organizing groups of relevant information. The Marine (also known as ArcMarine) data model available from Esri, is one of the more widely recognized data models for marine phenomena. The data model can be used to store several common data types: tables, marine points, marine lines, and marine areas (Figure 2-1).

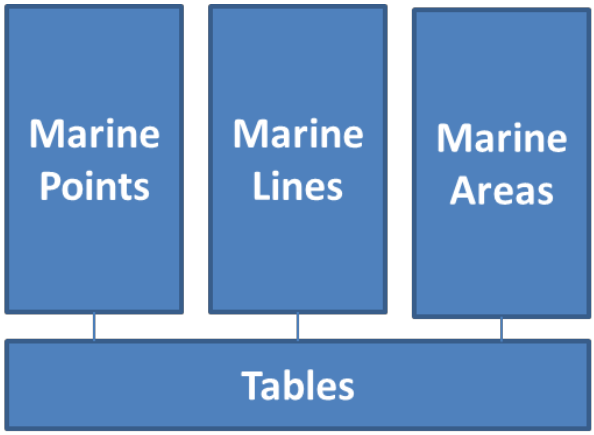


Figure 2-1: Common data types in the ArcMarine Data Model

The tables are intended to hold non-spatial elements. These elements are associated with one or more of the three spatial data types. Some of the associations are illustrated in Figure 2-2 between table elements and marine points and between table elements and marine lines. To demonstrate, a vehicle (non-spatial) such as a boat records a run line (spatial) when it is out on a trip. Similarly, a survey info (non-spatial) record can have one or more survey points (spatial) associated with it.

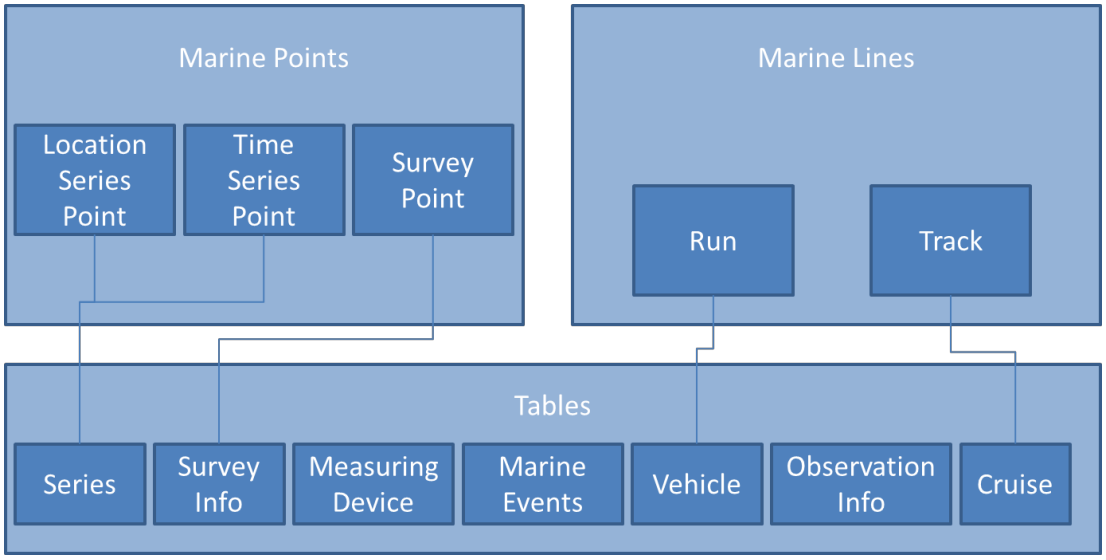


Figure 2-2: Some associations found in the ArcMarine Data Model

Whether data is collected in the field or downloaded from geospatial web services a data model, such as ArcMarine, is essential to managing the data. ArcMarine was designed to allow users access to analysis capabilities of GIS (Wright, Blongewicz, Halpin, & Breman, 2007). However, the data model is normalized and is not an ideal model for web and mobile projects. Additionally, the data model doesn't include a table associating the user who submitted the data. This demonstrates that it was not intended for storing volunteered geographic information.

2.2.2 GIS and Spatial Analysis

Once data has been collected and stored in a GIS, analysis can be performed to understand a wide array of topics. These topics include, but are not limited to human impact, and density and abundance. Some of the GIS techniques used to understand these topics include spatio-temporal analysis to understand patterns in the data and overlay techniques to understand human impact in marine mammals.

The use of space in understanding human impact is by no means a new idea, but it has become increasingly popular in the marine world since 2000. Since then, it has been used in a variety of applications. In 2008, Halpern et al. published a paper on research assessing the human impact on marine ecosystems. The study included several indicators relevant to fishing, pollution, and climate changes that were evaluated at a global scale. Raster analysis was performed to calculate and cumulate human impact indicators. The results found that there were virtually no ecosystems unaffected by humans. While this research is extremely valuable, similar research is needed at the local and regional levels.

Another research group recently released two studies using similar overlay and raster analysis techniques applied directly to marine mammals. One was directed at finding geographic ranges and patterns of richness and composition in an attempt at finding potential conservation sites (Pompa, Ehrlich, & Ceballos, 2011). Their second study was directed at identifying which species were at greatest risk and where the risk is

globally (Davidson, et al., 2012). A discussion of their findings found that more local data on migratory routes, locations of feedings, and calving/pupping could be used to produce better results. These are the exact types of data that this proof of concept was designed to collect, if fully deployed.

The preceding examples demonstrate the use of GIS in marine mammal research, and how it improves our understanding of a variety of topics. A need for larger scale assessment of human impact on marine mammals was revealed.

2.3 Understanding Web and Mobile GIS

Web and Mobile GIS are two different media for handling geographic information systems. The method of consumption is essentially the only difference between them. Web GIS applications are accessed through a browser on desktops and laptops. On the other hand, Mobile GIS applications are accessed through smart phones and tablets. However, advances in wireless communication have allowed users to access web-based GIS applications with both smart phone browser technology and native applications. This has caused significant overlap in the Web and Mobile GIS fields (Figure 2-3).

Web	Mobile
Connected	canConnect
Shares	canShare
hasLocation	hasLocation: LocationChanges
cloudStorage	localStorageBuffer
	hasOrientation

Figure 2-3: The overlap of Web and Mobile GIS

2.3.1 The differences between Web and Mobile GIS

There are significant advantages of Mobile over Web GIS. However, there are still significant technical challenges that give Web GIS an upper hand. Mobile GIS technologies can replace existing field paper-based workflows (Fu & Sun, 2011). Mobile GIS also provides a highly mobile environment that can be accessed by several users. Modern mobile devices are equipped with a GPS chip and other hardware devices that can reduce the amount of equipment needed in field work. Finally, Mobile GIS also enable users to work in a disconnected environment in which they have no access to internet or mobile services.

However, limited wireless communications also pose a technical challenge for the use of Mobile GIS. The desktop and server machines that are used in Web GIS provide more powerful CPU, memory, and battery power (Fu & Sun, 2011). Mobile GIS is also

limited in its screen size and keyboard size. This can be particularly inhibiting to field workers uncomfortable with mobile devices and in poor environmental conditions. There are significant advances of Mobile GIS over Web GIS. But because there are still some significant technical challenges, Web GIS still provides benefits.

2.3.2 Mobile GIS Approaches

The overlap in Web and Mobile GIS has resulted in two different approaches to developing Mobile GIS applications: native-based and browser-based. Native applications are those that are designed to run on a device's operating system, such as Apple iOS or Google Android. This forces the application developer to adapt the application depending on the operating system's platform language and operating system.

The two popular platforms are iOS using the Objective-C language and Android using Java. Objective-C is an exclusive language that isn't as commonly used (Viswanathan, 2012), while Java is a commonly used language and has extensive documentation for its SDK (Google, 2012). Java's large programming community has resulted in a reliable online resource community, easing the programming and cost burden for the project developer. Finally, phones with an Android operating system have become increasingly popular (Lloyd, 2012).

When developing a mobile application for Android devices, the developer must decide which operating system to develop for. To date, Gingerbread (2.3.3) is the predominant operating system on all Android platforms (Android 2.3.3 APIs, 2012). In the past eight months, all Android phones users had the option to switch to a newer operating system (IceCream Sandwich: 4.0.4) on their phones. However, eight months after IceCream Sandwich's release, only 10% of users had upgraded (Aguilar, 2012). This demonstrates that the developer must decide on both a platform and an operating system when creating mobile applications.

Just like many Web GIS applications, mobile browser-based applications can be built to work with a plugin, such as Flash Lite or Silverlight mobile, or with html and JavaScript (Fu & Sun, 2011). HTML browser-based applications can be developed once for a wider device range using a single language. This makes deployment of those applications across several platforms much easier. These types of applications also don't have to be purchased through application stores.

Unfortunately, HTML browser-based do not have full, and easy, access to the hardware on mobile devices, whereas native applications do. Additionally, because browser-based applications require connectivity, they cannot be used in disconnected environments. On the other hand, native applications can use local storage to work in such environments.

2.4 User Interface Design

Graphical user interfaces (GUI) have come to be an expected part of the experience for most computer users. When initially being developed, Apple Desktop Interfaces made two basic assumptions, "that the user can see, on a computer screen, what they are doing; and they can point at what they see," (Apple, 1987). Designing a good user interface requires the implementation of principles that have been repeatedly proven to be effective. Several sets of principles and rules of thumb have been put together by subject

matter experts, some of which include: simplify the structure of tasks, minimize memorization, plan for error, and know the user.

2.4.1 Simplifying Task

User Interfaces often provide new workflows for users to complete tasks that they were already doing using different methods. Asking users to change the way they currently perform a task can be difficult. In his book, *The Design of Everyday Things*, Donald Norman provides three technical approaches for ensuring success that are particularly relevant. The first is "Keep the task much the same, but provide mental aids." Mental Aids such as sticky notes and alarm clocks are simple examples of this. The second approach is, "Use Technology to make obvious what would otherwise be invisible." This approach can be implemented by giving the user feedback and allowing them to monitor the state of the system they are interacting with. The third approach is to "change the nature of the task." Having a thorough understanding of how the current workflow of a task operates, allows the designer to alter the way in which users are asked to provide input. Changing the nature of the task can make difficult tasks seems less daunting.

2.4.2 Reduce Memorization

There are several methods for reducing the memorization expected of the user. One method is to use a *see-and-point* method over *remember-and-type* (Apple, 1987). This can be implemented with the use of dropdown menus for making selections rather than text boxes for users to fill out. Another method is to use real-world metaphors. Doing this allows "users to transfer knowledge of how things should look and world," (Mandel, 1997). Using knowledge that is both in the world and in the head of the user, can make their experience with the user interface faster and more efficient (Norman, 1988).

2.4.3 Plan for Error

It is always safe to assume that the user will make mistakes in any number of ways when interacting with the interface. One method for handling this is to allow the user to recover if mistakes are made (Norman, 1988). This can be done by asking users to confirm actions before completing them and allowing users the permission to undo and redo actions.

2.4.4 Testing the Usability of Map User Interfaces

Understanding the usability of an interface is extremely important in its design and development. A direct method for determining whether or not the interface design is appropriate for the user is to test its usability. This is particularly important in user interfaces that are being designed for a broad audience, such as applications involving maps and the general public. A blog was released on mapbrief.com by Brian Timoney on how the public interfaces with local government web maps. The blog was based on research performed by a GIS Analyst with the City of Denver. There were a couple of findings they were particularly relevant. The first was that people rarely changed default settings, this includes the default basemaps. Giving the user too many options may

overcomplicate a normally simple task. The analyst also found that using an auto-complete in the map's search box drove clean queries (Timoney, 2012).

A study was recently performed on the usability of a citizen science web application. The researchers found that users had a difficult time understanding the concepts of layers in the map's legend, and that layers could be turned on and off (Newman, Zimmerman, & Crall, 2010). This demonstrates the importance of knowing the user and the power of the real world metaphors method to engage them.

2.5 Summary

This chapter reviewed the background information relevant to the project. Section 2.1 discussed VGI in Web and Mobile Applications. VGI is a relatively new field, and its uses in scientific research are extremely limited. Section 2.2 talked about the GIS-based methods used in the marine field for sharing and storing data. Geospatial web services are being used to share data at large and small scale, and the use of geodatabases and data models can be useful is organized data for analysis. The use of GIS in marine spatial analysis was also discussed in this section. There is a need for large scale assessment of human impact on marine species and their environments. Section 2.3 described the differences between web and mobile GIS and why each of them plays an important role. While mobile GIS simplifies field-based workflows and can be accessed by a wide audience, web GIS solutions do not have memory, battery life, or connectivity issues. Section 2.4 discussed user interface design principles and demonstrates the importance in understanding and applying them to web and mobile applications is extremely important.

Chapter 3 – Systems Analysis and Design

This chapter discusses the analysis of the system design of the proposed solution. More specifically, it includes a Section 3.1 which revisits the problem statement. The problem statement was used in a requirements analysis. The results from this are presented in Section 3.2. The requirements analysis was used to develop a system design and project plan. The proposed system design and the plan for its implementation are described in Section 3.3 and 3.4. The chapter is wrapped up in a summary section.

3.1 Problem Statement

The challenge that the client, Dr. Lei Lani Stelle, faced was how to generate a high volume of quality data for her long-term study on marine mammal migratory behavior and human impact stored in a well-organized fashion for analysis. In order to reduce the cost and effort required in the data collection process, Stelle decided to explore a VGI approach to collecting this data, which incorporates the collection of data by both researchers and members of the general public.

3.2 Requirements Analysis

Functional requirements describe the information and answers that the system will provide to its end users. For example, a functional requirement of the system was to allow users to submit marine mammal sightings from both a mobile device and a website. On the other hand, the non-functional requirements describe the way in which the system should perform and includes technical, operational, and transitional requirements. The technical requirements describe both the technology the client will need to maintain the system and the technology the end user will need to access the system. The operational requirements include the day-to-day or periodic maintenance requirements that the client will be responsible for to keep the system up and running. Finally, the transitional requirements are those needed for handing the system over to the client and end-users, such as training or usage documentation. The following subsections describe in detail the functional and non-functional requirements of the system that are outlined in Table 3-1 below.

Table 3-1: Functional and Non-Functional Requirements

Functional Requirements	
Store survey logs and corresponding events, including sightings and their corresponding evidence (such as photos), and position updates	
Allow users to submit events and corresponding evidence via mobile or web application	
Tie submitted events with a particular user	
Download of queried data on web application	
Basic visualization on map of submitted data points	
Allow users to rate their confidence in their submitted observations	
Automated position updates on mobile application	
Data stored locally on mobile device for future offline development	
Non-functional Requirements	
Technical Requirements	User interfaces and application navigation for both the web and mobile applications
	Esri's ArcGIS Desktop 10.0
	Esri's ArcGIS Runtime SDK for Android
	Esri's ArcGIS API for JavaScript
	Notepad and Aptana 3
	Eclipse Integrated Development Environment
	Esri's ArcGIS Server 10.0
	Android Software Development Kit
	Internet Information Services (IIS) Web Server
	Domain Name
Operational Requirements	Archiving the geodatabase since users are editing the default tables
	Monetary costs for operating and maintaining the web and mobile applications
Transitional Requirements	Distribution of the mobile application

3.2.1 Functional Requirements

There were several functional requirements of the system, as outlined in the table above. The system had to be capable of storing survey logs from a user's trip and all of the events that occurred in a single trip. There were two types of events that occurred during a trip. The first was a position update, which included a time and position. The second was an observation (or a sighting). Sightings consisted of time and position, but they also had information about the species observed and relevant information. This included a confidence rating on the data being submitted. The system was designed to classify all incoming records by subtype. If the incoming event was a sighting, the observation is classified as one of four marine mammal categories. If the incoming event was a position update, it gets recorded accordingly.

The system allowed users to submit the above mentioned survey logs and corresponding events using the web and mobile applications. This was done by developing a form that could be accessed by both. The form was developed to allow users to associate a photo with the observation being submitted. The data associated with the events are stored locally on the mobile device and the events are submitted to the geodatabase through the use of feature services. This was done using a creating a table in a SQLLite database on the device itself.

The events submitted can be associated with the user that enters them. In the web application, the user must log in before submitting any observations. The username is temporarily held so that it can be used to tie observations with the user that recorded them. On the mobile device, the user can associate their data with an email addresses on their phone. Alternatively, by default, all data is associated with an "anonymous" user account.

The web and mobile applications were developed to allow users to visualize recorded observations and position updates. Visualization on the web application was made possible through the use of map services that access the data in the geodatabase. The events recorded on the mobile device are rendered locally from the data stored on the device rather than through map services. The feature services are only accessed during the submission of an event from the phone.

There were also functional requirements specific to the device. Additional core requirements for the web application were that users had to be able to query and download data. Querying of the data was made possible through the use of toggle buttons and a date range that the user could adjust accordingly to visualize their desired results. The results from the query are used if the user decides download the data from the website. In order to download data, the user had to use a single button. This button is associated with a geoprocessing service that selects the desired results and creates a zipped up shapefile that is sent to the user's temporary folder on their computer.

There was also a final functional requirement for the mobile device: position updates had to be required automatically for the user. This was completed by using an Android function that would repeat a particular set of tasks every given amount of time. The repeated tasks recorded a new observation in the local database, submitted the observation to the geodatabase using feature services, and rendered the new position update on the map.

3.2.2 Non-Functional Requirements

The technical requirements are the backbone of the non-functional requirements as they are focused on the technologies required to build the system. Web services were created from a geodatabase and published from within ArcGIS Desktop 10.0 to ArcGIS Server 10.0. These services were made available to users through the mobile and web applications.

The web application was composed of graphical user interfaces and application navigation. It was hosted using Internet Information Services (IIS) web server and required a universal resource locator (URL) for access. The web application must be accessed using Google Chrome or Mozilla Firefox. The application was developed using Notepad and Aptana environment and utilized Esri's ArcGIS Application Programming Interface (API) for JavaScript.

The mobile application was also composed of graphical user interfaces and application navigation. It was developed for the Android phone with an operating system of 2.3.3 or lower in the Eclipse Integrated Development Environment (IDE). The mobile application utilizes the Android Software Development Kit (SDK) and Esri's ArcGIS Runtime SDK for Android.

In order to fully understand both the operational and transitional requirements, it is important to note that the system was built as a prototype, or proof of concept, the client

intended to use to get funding for development of a more permanent system. Given this, the prototype system was intentionally designed to have very few operational requirements, and this was done at the expense of security. This means that users were to edit the default data set. There was no versioning or automatic archiving of the data. It is the client's responsible to periodically download and archive the dataset. The client will also be responsible for the annual/semi-annual monetary requirements of maintaining a GIS server, web server, and website.

The system was developed for the general public, so there was no documentation or training made available to the client for using the mobile and web applications. The only other transitional requirement was the distribution of the mobile application. While this could be available to the public through Android's App Store, this was beyond the scope of the project. The client is responsible for personally downloading it onto the devices that will be using it.

3.3 System Design

The design of the system was guided by the system requirements described above. This section describes the system architecture design for both the mobile and web applications, along with constraints on the design. From this point forth, the system will be broken into two categories: server side and client side (Figure 3-1). This is done to clarify the difference between activity on the user side of the solution and the server side, where the system resides. The client side of the application sends requests to the server side, which then sends responses back to the client side. The server side is composed of the web server, ArcGIS Server, the application, and a geodatabase. The client side is split between the web component and the mobile component.

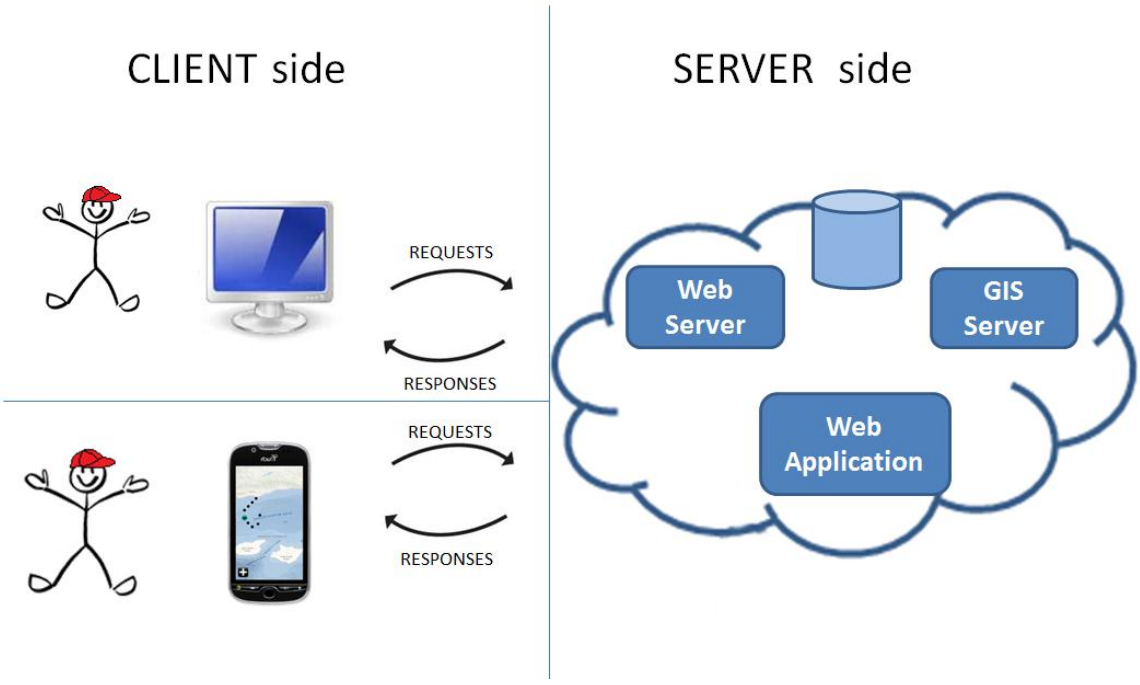


Figure 3-1: Overall system design

3.3.1 Web Application

The web application was designed as a shell for future development of the application. It sits on the client side and consists of a user interface, along with supporting JavaScript functions. The JavaScript functions are responsible for the dynamic capacity of the web application and were developed using the DOJO JavaScript framework and the ArcGIS API for JavaScript. Both the JavaScript framework and the API were consumed by the application through services.

The user interface includes a *Home*, *Map*, *My Observations*, and a *Learn* tab (diagrammed in Figure 3-2 above). The *Home* and *Learn* tab were not developed during the scope of this project. The *Map* tab was designed for use by visitors that didn't want to log into the system. The map tab was designed to allow the user to visualize observations from a map service on top of Esri's Ocean basemap and query them by date and species type. Upon clicking on a particular observation, the application was designed to display the attributes and photo of the selected observation in a popup window. The *Map's* JavaScript methods are responsible for ensuring the queries are sent to the server and that observations are rendered on the map.

The *My Observations* tab requires users to log in before access. Just like the *Map* tab, this one was designed to also allow users to visualize and query the data, the difference being that the users are provided with additional query options.

The advantage of the *My Observations* tab is that users can download data after querying the database and after visually confirming the selection criteria they wanted. The *My Observations* tab was designed to allow users to submit new observations. This is completed using a feature service. Just like the *Map* tab, the dynamic functionality and server communication are handled by JavaScript methods in the *My Observations* tab.

There were several design constraints to take into consideration of the system design. The web application was designed to only work with a modern browser, such as Firefox or Chrome. Modern web browsers support a combination of standards, while earlier browsers only support very simple HTML standards. Additionally, the web application was designed to work with the ArcGIS API for JavaScript. This API was chosen because of its flexibility and popularity in the web programming field. It also is the only client-side web API that doesn't require a plugin for the browser.

3.3.2 Mobile Application

The mobile application has both user interface and supporting components. There are three components to the user interface design of the mobile application: the *Home* activity, the main activity called the *Survey Map*, and the *Observation* Activity. The supporting components on the mobile application's client side were the *Local Database* and the *Events Database Manager*.

The *Home* activity was responsible only for starting a trip and allowing users to decide which of their email accounts they wanted to associate their survey log with through a login window. The *Survey Map* was responsible for allowing users to visualize their trip. The activity also had a method for automatically recording and displaying the user's position every five minutes. In addition to recording the position update, the *Survey Map* added a new position marker to the map. Finally, the *Survey Map* was responsible for adding observations stored locally to the map. The *Observation* interface

provides the user with a form to complete with a series of dialog boxes with necessary information.

There were multiple design constraints relevant to the client side of the mobile application. The first was the choice between a native and web application. The difference between these is discussed in more detail in Chapter Two. A native application was chosen for the system because one of the functional requirements was storing the data locally on the device. Having this workflow in place reduced future development burden of the application.

The second design consideration for the mobile device was the platform for which it would be developed natively. The Android platform was chosen because of Java's popularity and extensive documentation. It was also chosen because it is the operating system on more mobile devices than the completing Apple iOs platform. The application was developed for the Gingerbread operating system. The choices for the mobile platform and operating system were discussed in Chapter Two.

3.4 Project Plan

The project plan and implementation was similar to spiral and agile models for software development (Figure 3-2). After a requirements analysis, three separate phases were initiated: the geodatabase, the web application, and the mobile application. Each phase had the following steps: design, development, test, and discuss.

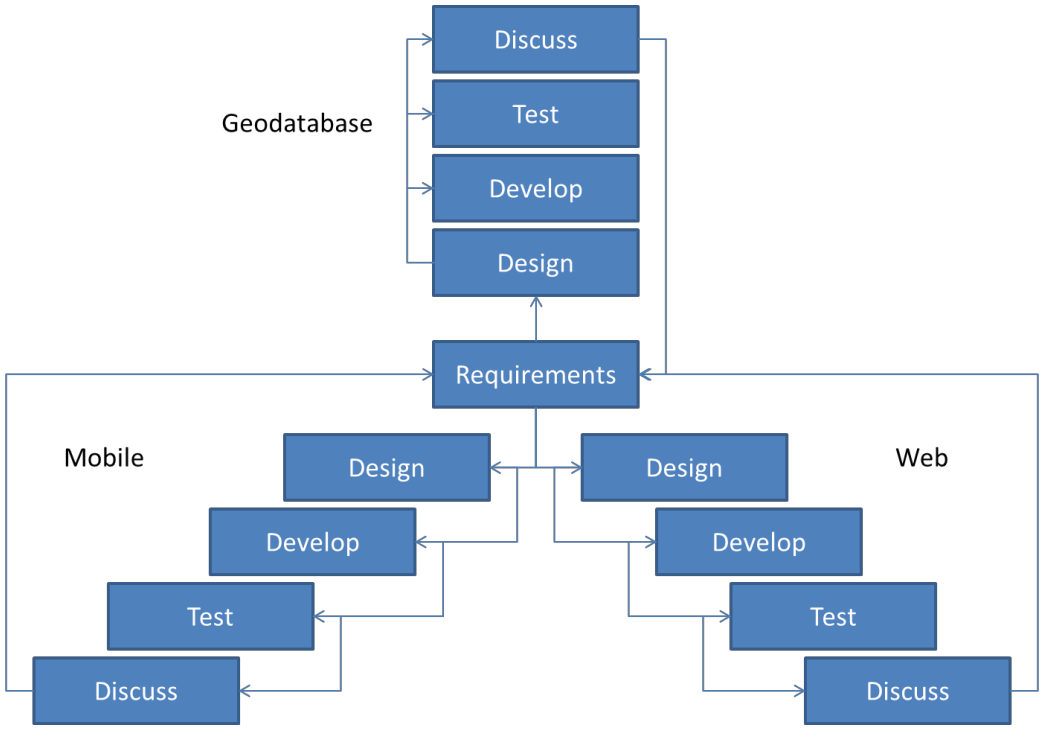


Figure 3-2: A model of the project plan

During the design task for the geodatabase, the client's functional and non-functional requirements were incorporated into a logical model. During every successive design task for both the web and the mobile applications, a mockup of the user interface was

created and approved by the client. During the design and development stage for the web and mobile applications, the developer determined the appropriate functions that would be used in development through intensive research and training on topics including: the ArcGIS API for JavaScript, the Android Architecture, the Eclipse Integrated Development Environment, the Android SDK, and the ArcGIS Runtime SDK for Android.

During the development phase of the geodatabase, a logical model was developed. A logical model was created for both the client side (mobile applications) and the server side (geodatabase) of the system. Additonally, map and feature services were created. The development of the web and mobile applications were dependent on the completion of the geodatabase development.

Testing of the geodatabase and the client side applications were performed simultaneously. This was done because the web and mobile applications consumed services created from the geodatabase. In addition to the developer, thhe client was also asked to test the applications. After passing these tasks, a prototype was complete for each phase. During this time, a dialog was exchanged between the developer and the client. The client provided feedback on both the user interface and the functionality of the applications. Planning for the next prototype was then completed.

During the project development life cycle, some modifications were made. Testing the mobile application in the field gave insight, resulting significant modifications. For example, the original project plan was for complete offline editing on a mobile device. Field testing demonstrated that there was full connectivity in a region that could be used as the study area. Because of this, the mobile application no longer became a complete offline application. The application was designed to consume online basemaps rather than cached maps stored locally on the phone. However, the developer chose to continue storing the data locally on the device. This way, the only demand for mobile connectivity was when the user wanted to sync observations.

Additionally, the original plan included user authentication upon login and had the possibility of different permissions for different user types. Specifically, user authentication was planned to be done using openID (Google accounts), so all users must have a Google email account. However, because of the requirements of the server, this portion of the solution will not be implemented due to University regulations. Therefore, user verification was dropped from the implementation plan. Varied permissions were not incorporated into the applications, but their capability was demonstrated within the geodatabase.

3.5 Summary

The system requirements analysis process resulted in a system design that met the needs of the client. This chapter revisited the problem statement, discussed the requirements analysis and found that there were nine functional requirements and 13 non-functional requirements. In addition to the requirements analysis, the system design was also presented. This started with an overall description of the system design and continued with the design of the two client-side applications. Finally, the chapter concluded with a discussion of the project plan. The plan consisted of five major tasks and was adjusted throughout its implementation. With an understanding of the required functionality and system design, the database was ready to be developed.

Chapter 4 – Database Design

The purpose of this chapter is to describe the data models used in the project solution. Section 4.1 describes the conceptual model, which was used to determine the classes and attributes needed to store the data required for the project. Section, 4.2 describes how the conceptual model was altered to balance data duplication and performance optimization in the logical model. Section 4.3 describes the data sources, and Section 4.4 describes the methods performed on the data before they were ready for use in development of the web and mobile applications.

4.1 Conceptual Data Model

The conceptual database model was developed through conversations with the client and the field data collection sheet (described further in 4.3). The conceptual model helped define the necessary database classes needed to solve the client's problem (Figure 4-1) These are the primary classes (shown in italics) that needed to be defined: *Observer*, *Survey*, and an *Event*. The relationship between the *Observer* and the *Survey* is that an *Observer* records surveys. E+9*-ach *Survey* contains one of more events. The *Event* table holds two different types of events: position updates and observations. All events must contain time and location. An event of the Observation type contains additional information. An Observation event may contain evidence, such as a photo. The *Evidence* class is the evidence, such as a photo, associated with a particular event.

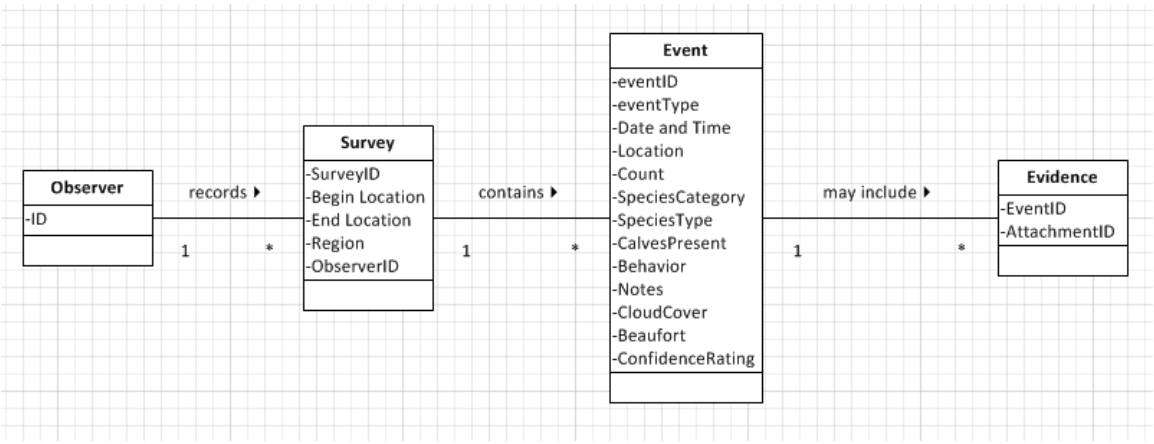


Figure 4-1: Conceptual Model

4.2 Logical Data Model

While the conceptual model is normalized and describes the client's needs abstractly, the logical model describes how the database schema was designed. The logical model design took into consideration the most appropriate way to store data for use in web and mobile-based applications. A diagram of the logical data model can be found in Figure 4-2 below. There are two extremes in data modeling: a completely normalized model and a flat file model. Flat-file databases have no explicit relations between tables, while normal databases have undergone a normalization process to eliminate data duplication

and minimize the use of space. The advantage that flat-file databases have over normalized ones is that they perform much faster. This is desirable in databases used for web applications. The flat-file approach also reduces the learning curve for new administrators of the database.

Because of this, the *Observer* and *Survey* classes were consolidated into the *Event* table. This resulted in a geodatabase with only an *Event* table. It is important to note the consequences of using a flat-file database over a normalized one. There is significant data duplication in flat-file database and there are none in normalized ones. Additionally, the compartmentalized effect of normalized databases makes them easier to maintain and update. The use of domains and subtypes were incorporated into the database design to reduce maintenance demands. A complete listing of these can be found in Appendix A.

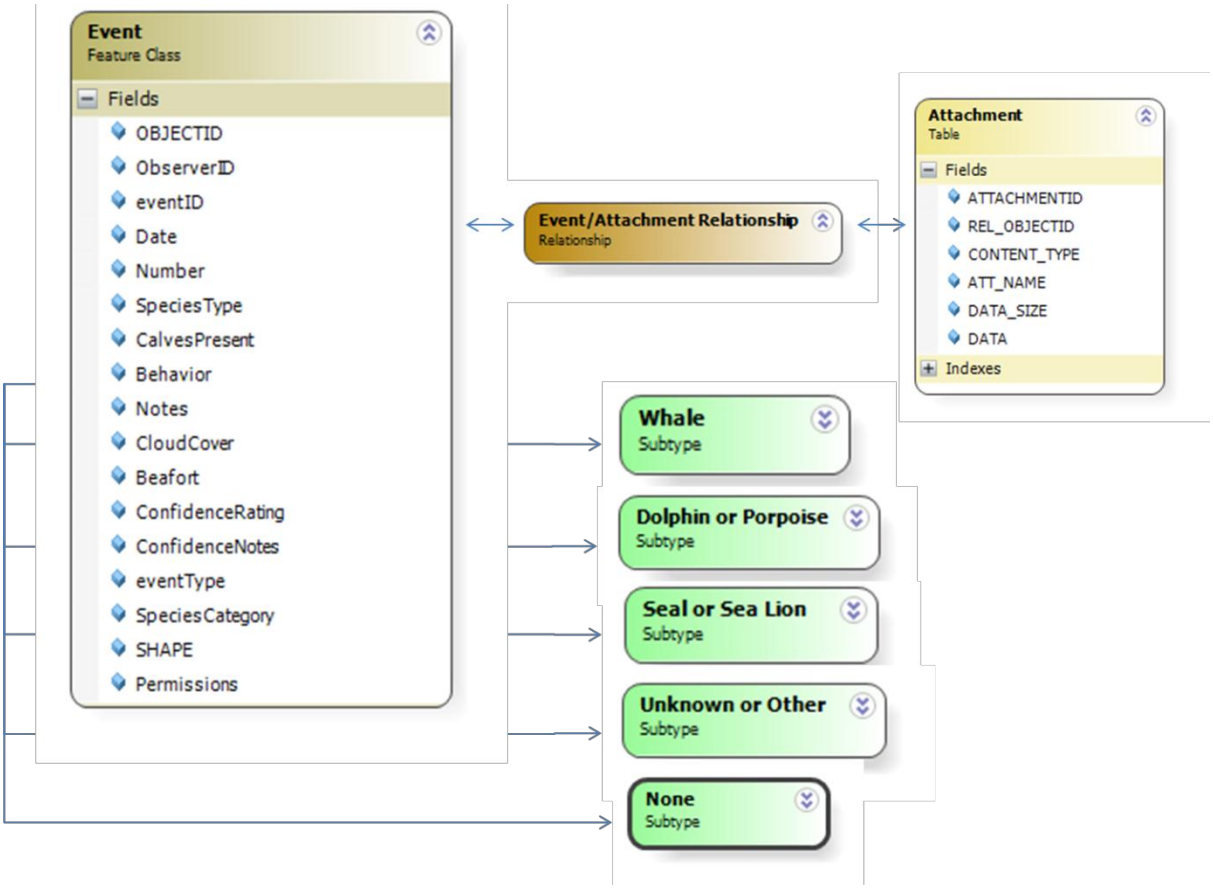


Figure 4-2: Server side logical model diagram

The data needed to be stored on both the local client mobile device and the server, so two different components to the logical model were created. The *Event* table was stored as a feature class in an ArcSDE geodatabase on the server side and it was stored as a table in a SQLLite database table on the mobile device. This is outlined in Figure 4-3 below.

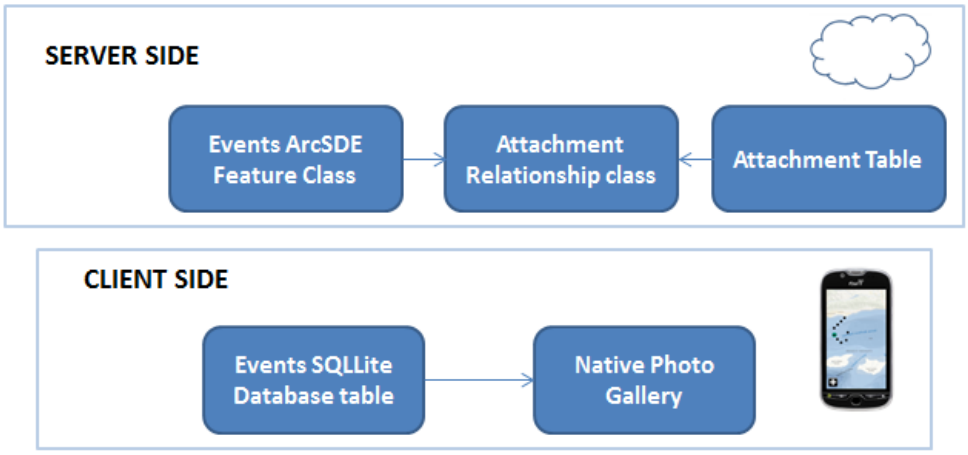


Figure 4-3: Client side vs. server side database structure

The primary difference between the way the data were designed to be stored in the logical model has to do with the way that evidence (photos) are associated with the events in the *Event* table. There are several methods for associating evidence, such as photos, with records in a table. These methods include storing a hyperlink as a text field, storing the evidence as a blob field, or storing the evidence in a separate attachment table. For the server side *Events* feature class, an attachment table was chosen as the method for storing the evidence because it is not dependent on path names internally or URLS on the web. Additionally, it keeps the evidence stored within the geodatabase without being queried every time the event record is queried. This increased the speed of queries, which is essential in web applications.

The photos were stored differently on the client side. The physical photos were stored in the phone's native gallery. Only a pathname was stored in the *Events* table. The only other difference between the server side and client side *Event* tables was that the client side table contained an additional field for tracking whether or not the records have been rendered on the map.

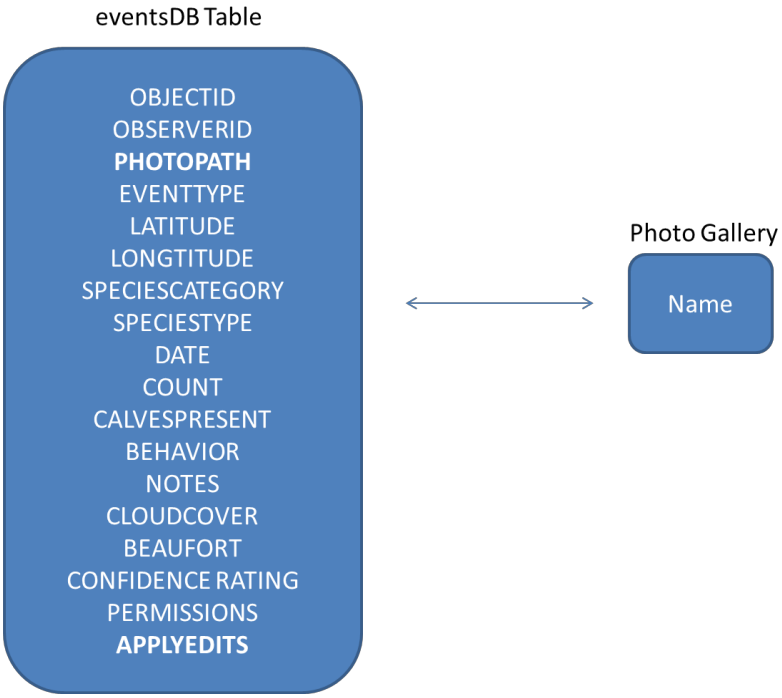


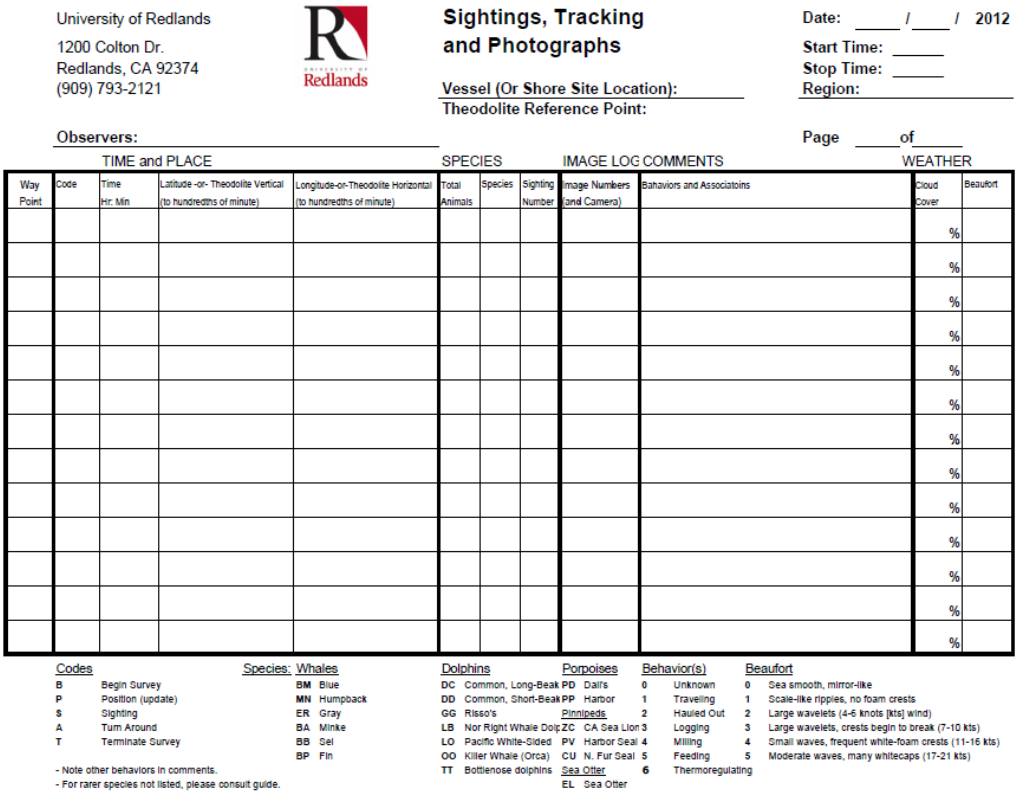
Figure 4-4: Client side logical model diagram

4.3 Data Sources

Recall that the project's mission was to build prototypes for collecting and managing marine mammal sightings. The development of the proof of concept prototypes didn't require much data for ensuring that the client's needs were met. Because of this, the data could have been synthesized.

Rather than creating a complete synthetic dataset, data were obtained from the Channel Islands National Marine Sanctuary (CINMS). CINMS has collected marine mammal sightings since 2003 and provided an Excel sheet with nearly 16,000 marine mammal sightings. Each sighting contained latitude, longitude, and a handful of other attributes, including: mammal category, type, date, vessel, location, count, and behavior.

In addition to the CINMS dataset, the data collection sheet used in the field was provided by the client. This data collection sheet is the form used by students and volunteers in the field (Figure 4-5). It was used to provide insight for the development of the conceptual and logical models. It was also used for development of domains for fields.



University of Redlands
1200 Colton Dr.
Redlands, CA 92374
(909) 793-2121

Sightings, Tracking and Photographs

Date: ____ / ____ / 2012
Start Time: _____
Stop Time: _____
Region: _____

Vessel (Or Shore Site Location): _____
Theodolite Reference Point:

Observers: _____

Page ____ of ____

TIME and PLACE					SPECIES			IMAGE LOG	COMMENTS	WEATHER	
Way Point	Code	Time Hr: Min	Latitude -or- Theodolite Vertical (to hundredths of minute)	Longitude-or-Theodolite Horizontal (to hundredths of minute)	Total Animals	Species	Sighting Number	Image Numbers (and Camera)	Bahaviors and Associatoins	Cloud Cover	Beaufort
										%	
										%	
										%	
										%	
										%	
										%	
										%	
										%	
										%	
										%	
										%	
										%	
										%	

Codes
B Begin Survey
P Position (update)
S Sighting
A Turn Around
T Terminate Survey

Species: Whales
BM Blue
MN Humpback
ER Gray
BA Minke
BB Sei
BP Fin

Dolphins
DC Common, Long-Beak
DD Common, Short-Beak
GG Risso's
LB Nor Right Whale Dolp
LO Pacific White-Sided
OO Killer Whale (Orca)
TT Bottlenose dolphins

Porpoises
PD Dall's
PP Harbor

Pinnipeds
ZC CA Sea Lion
PV Harbor Seal
CU N. Fur Seal

Sea Otter
EL Sea Otter

Behavior(s)
0 Unknown
1 Traveling
2 Hauled Out
3 Logging
4 Milling
5 Feeding
6 Thermoregulating

Beaufort
0 Sea smooth, mirror-like
1 Scale-like ripples, no foam crests
2 Large wavelets (4-6 knots [kts] wind)
3 Large wavelets, crests begin to break (7-10 kts)
4 Small waves, frequent white-foam crests (11-16 kts)
5 Moderate waves, many whitecaps (17-21 kts)

- Note other behaviors in comments.
- For rarer species not listed, please consult guide.

Figure 4-5: Data collection sheet used by client's students and volunteers

4.4 Data Scrubbing and Loading

Prior to loading the data provided by CINMS into the server side geodatabase, the geodatabase schema were developed. The schema included an *Event* feature class. The fields for the *Event* feature class included all of those listed in the logical data model in the preceding section. Domains for the feature class were created using the data collection sheet described in Section 4.3. Additionally, subtypes were created for four different species categories (Whale, Dolphin or Porpoise, Seal or Seal Lions, Unknown or Other). An additional category of "None" was created for position updates that had no observation data associated with them. The species categories were chosen for the subtypes to simplify the rendering of symbols and number of layers needed in the map service.

Prior to loading the CINMS dataset into the *Event* feature class, the longitude values were adjusted to a xy grid so that they would be stored in the correct hemisphere. Additionally, the CINMS was adjusted to include values for the coded values domains. Approximately 100 of the CINMS data points were imported into the geodatabase. Once in the geodatabase, several of the fields were either populated with synthetic values or left blank. The fields that were synthesized include: Observer ID, event type, date, event ID, calves present, cloud cover, beaufort, confidence rating, and notes.

The final step in prepping the data to be used in development was associating an attachment table with the events feature class via a relationship class, which was created.

A photo was attached to each of the records in the *Event* feature class with an Observation subtype. The photos used were taken by Earthwatch volunteers during a whale watching trip.

4.5 Summary

This purpose of this chapter was to describe the database model and the data that were relevant to the project development. In the section on the conceptual model, the classes and their associated attributes that represent the problem proposed were described. The conceptual model consisted of four classes. The logical model was then discussed that was made during development of the solution for faster access of the data in both the client and server side environments. It consisted of two classes. The chapter also discussed the sample observation data provided by CINMS. It concluded with a discussion on the schema that was built and the scrubbing that was performed on that data prior to loading them into the geodatabase. With a well-designed database, and sample data ready for testing, the applications were ready to be built.

Chapter 5 – Implementation of the Web Application

Two client side components were developed to meet the client's requests: a web application and a native mobile application. The mobile application was designed for users who are on personal or chartered whale watching boats and interested in tracking their trip and recording their observations instantly. On the other hand, the web application was designed for users who are either interested in visualizing the data as a visitor, or submitting single point observations and downloading the data for analysis as a logged-in user. This chapter describes the implementation of the web application. Section 5.1 discusses the web application's user interface. Section 5.2 discusses the functional components of the web application. The chapter concludes in Section 5.3 with a summary of the web application.

5.1 Web Application User Interface

The web application serves a variety of roles and was organized into four tabs: Home, Map, My Observations, and Learn (Figure 5-1). The Home and Learn tabs were completely undeveloped and acted as placeholders for future work, as requested by the client. The Map tab and the My Observations tab were divided into a content pane for a map object and accordion panes for the tools and forms available to the user. The accordion pane is a web control object that is displayed on the user interface. It appears as a container for holding web content. The map content pane is displayed to the user during the entire session, whereas accordion panes can be hidden and displayed as desired.

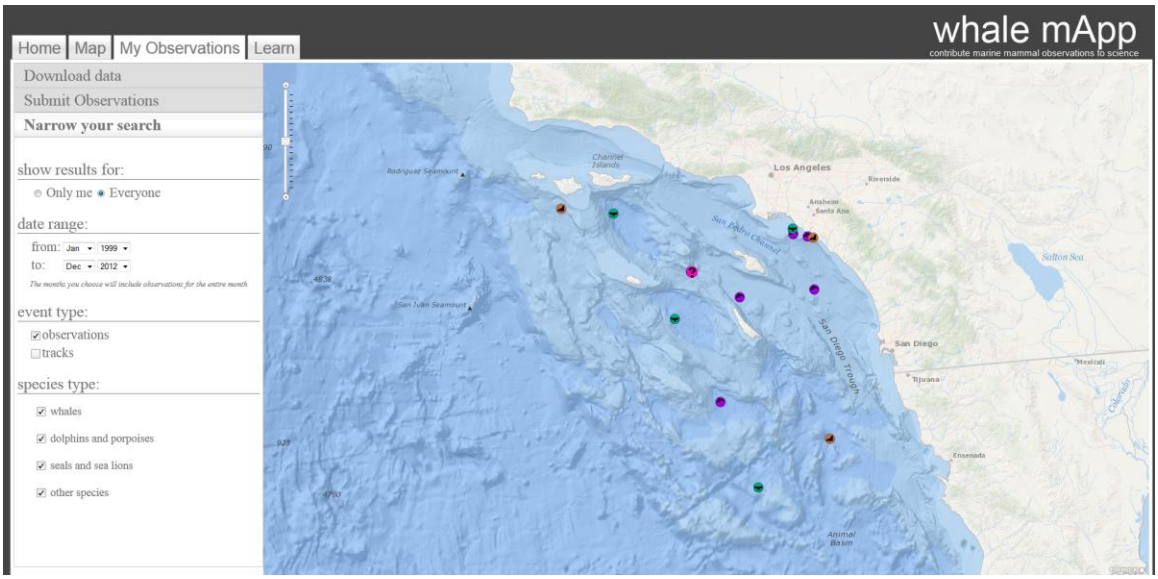


Figure 5-1: Functional components of the web application.

5.1.1 The *Map* tab

The Map tab was designed for the users who don't wish to log into the system and are interested in visualizing observations, querying the database, and identifying details on selected events. It consists of several components (Figure 5-1), one of which is a map for visualization. The map is the central component of the tab. The tab includes an identify popup tool that allows the user to learn information about a particular observation. The tool triggers a popup when the user clicks on a symbol within the map. The identify popup tool was designed to be accessed only when a user clicks on an observation in the map object.

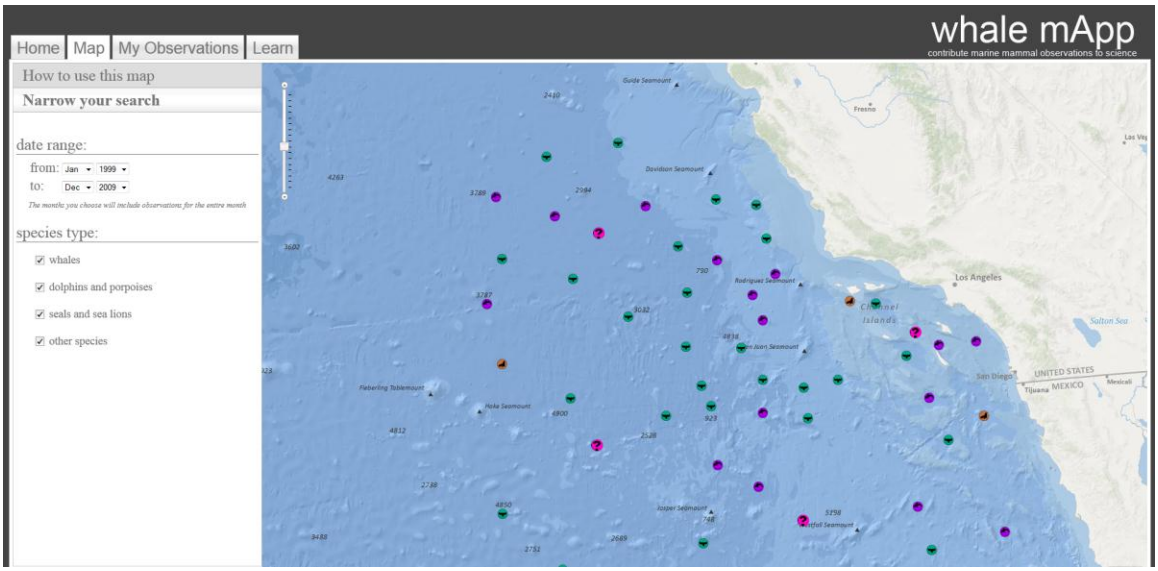


Figure 5-2: Overview of the *Map* tab of the web application

The user interface was also designed to include basic querying tools. Querying tools select the chosen records and only displays them to the user. These basic querying tools are made available to the user through a content pane on the left side of the tab. The date range query tool allows the user to query by a "to and from date" using drop down boxes for month and date. These date tools are illustrated in Figure 5-3.

Figure 5-3: Date Query Tool

The user can also query by species type. Species type options are made available to the user through checkboxes (Figure 5-4). This allows the user to view results of any combination of species at the same time.

Figure 5-4: Species type query tool

5.1.2 The *My Observations* Tab

The My Observations tab was designed for an active user interested in a more advanced interaction with the data (Figure 5-5). When the user accesses the tab, they are prompted to log in. Upon log into the system, the user has access to the same map, and identify popup tool as in the Map tab.

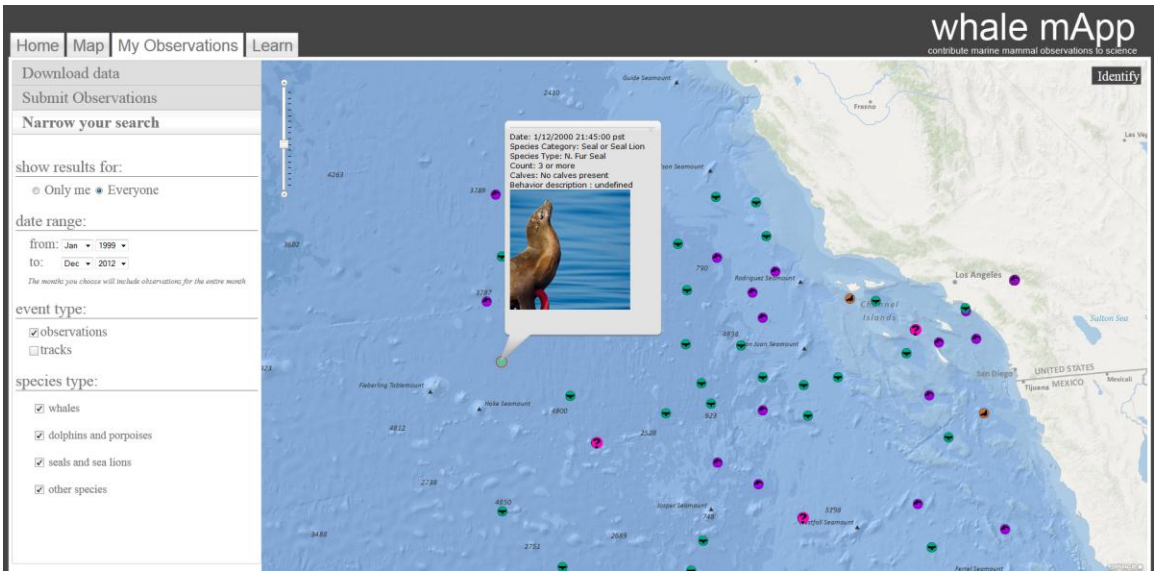


Figure 5-5: My Observations tab

This tab has more advanced query tools which are displayed within an accordion pane. The user can query by event type, allowing them to see both observations and tracks. This query tool is made available to the user as checkboxes, allowing them to view any combination of event types (Figure 5-6).

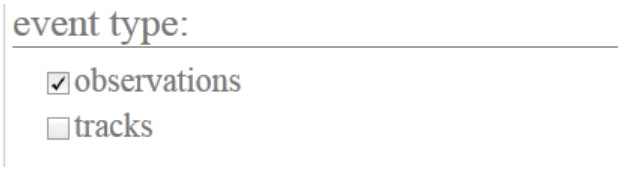


Figure 5-6: Event type query tool

They can also choose to see only their observations or the observations of all users. These options are made available to the user through a radio button, and they can choose only one option or the other (Figure 5-7).

Figure 5-7: Show results query tool

Additional accordion panes were designed for additional tools, one of which is the download data tool, which allows users to download data as a shapefile after they have queried the data with the desired attributes (Figure 5-8).

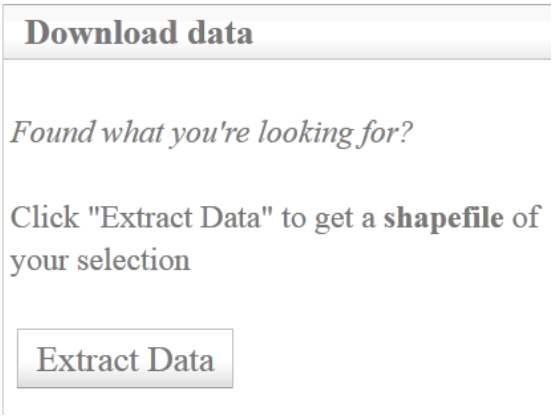


Figure 5-8: Download data tool

Also within an accordion pane, the user has access to a Submit Observations Form (Figure 5-9). This form contains all of the information required to submit a new observation, along with the button needed to submit it. Most of the form's components were made from Dojo form widgets, also known as Dijits (Dojo, 2011).

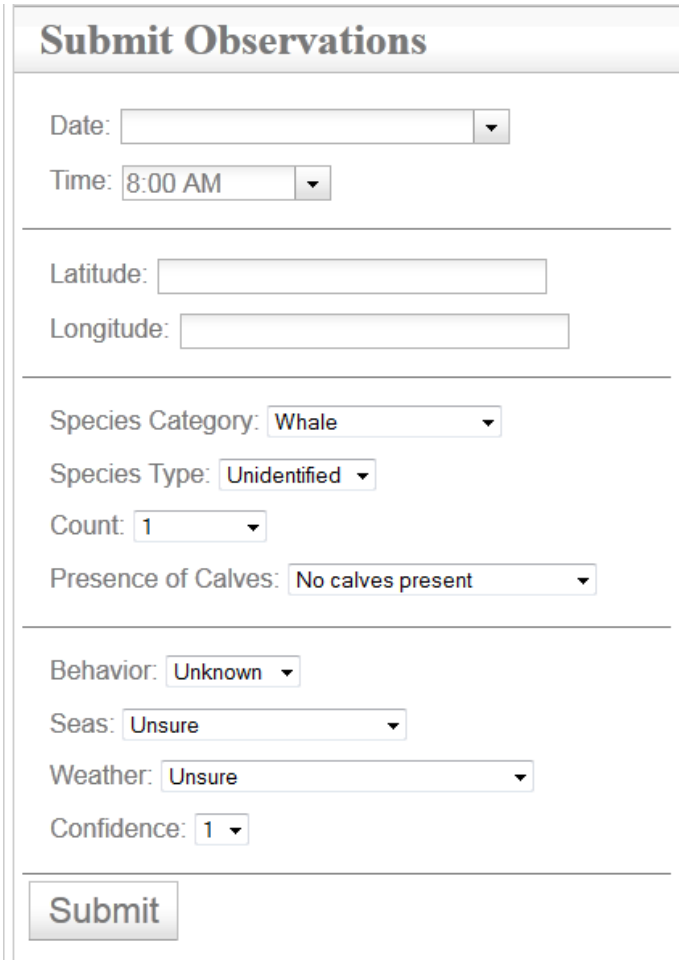


Figure 5-9: Submit observations for in accordion pane

The Submit Observations tool was designed to minimize the amount of error in data entry. This was done using drop-down lists and auto-fill for every option other than location (Figure 5-10). A NumberTextBox (a Dijit) was used to restrict the user to only entering numbers in the locational text boxes.

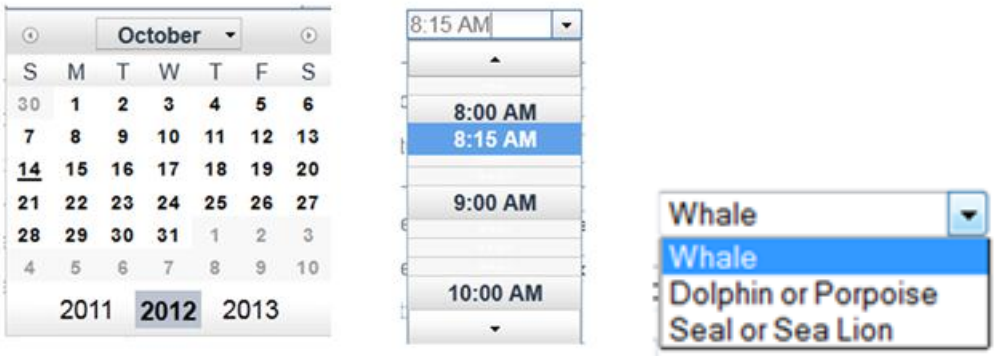


Figure 5-10: The use of auto-fill and drop-down use for the Submit Observations form

5.2 Functional Components

The user interface components described above are dependent on several functional components. The purpose of this section is to describe each of the functional components in detail. The functional components and their corresponding classes and methods are illustrated in Table 5-1. Classes are shown in lower camel case (`ExampleClassName`) and methods are shown in lower camel case (`exampleMethodName`).

Table 5-1: Functional components and their corresponding methods

Functional Component	Class or Method
The Map	`Map`
Identify Popup Tool	`identify`
Query Tools	`dateChanged, speciesChanged, changeLayerDefinition, eventTypeChanged, showResultsFor`
Download Data Tool	`downloadData`

5.2.1 The Map

As mentioned, the map appears in both the Map and My Observations tabs. The map's content is dynamic, depending on the extent, map layers, and query functions. The map extent was set to (in meters):

- -13,252,801.136300 (minimum x)
- 4,388,898.711300 (minimum y)
- -1,4091,002.8127 (maximum x)
- 3,508,363.61690 (maximum y).

The layers contained within the map component are outlined in Table 5-2 below. The map, feature, and geoprocessing services that the map layers were composed of were published through ArcGIS Server from a map document within ArcGIS Desktop.

Table 5-2: Web application layers

Layer Name	Type	Usage	Mode
Ocean Basemap	Basemap	Basemap	NA
Events	Feature Service	Display of all observations	Selection
Attachments	Map Service	Table of image attachments	NA
ExtractEvents	GP Service	GP tool	NA

5.2.2 Identify Popup Tool

When the user clicks on an observation on the map document, the `identify` method is called. A query is sent to the server for any feature within the spatial extent of the envelope. If something is returned, a new selection symbol is defined for the events feature layer. The layer definition is reset (using the `Map` object's `setLayerDefintion` and `setSelectionSymbol` methods) with the new selection and selection symbol. This replaces the species category symbol with the unique selection symbol (Figure 5-11). This unique selection symbol has a green fill and red outline.

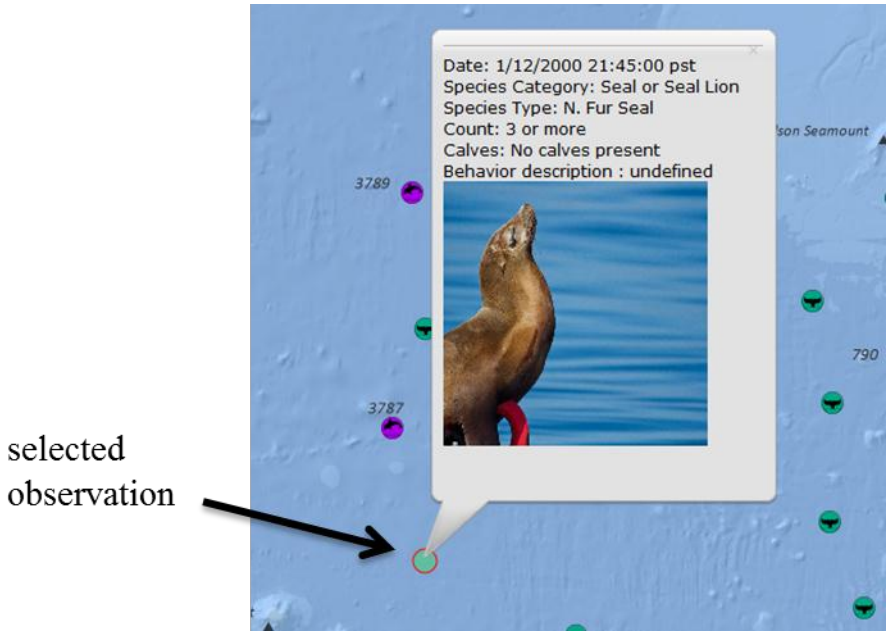


Figure 5-11: Symbol selection and popup window.

After the observation is assigned a new observation symbol, a second query task is defined and executed which searches the attachment table for a picture corresponding to the selected observation. If an image exists in the attachment table, the content of the map's infoWindow is updated, the window is resized, and the infoWindow is displayed.

5.2.3 Query Tools

The methods that perform the query are triggered by listeners on each of the query tool options. A listener is an object that performs an action when the user interacts with its corresponding part of the user interface. Each listener calls a respective function that builds a string to query its corresponding attribute in the feature layer. For example, when the user changes the date in the query tools, the `dateChanged` method is called. Afterwards, the `changeLayerDefinition` is called which pieces together the appropriate string consisting of each query tool's current value. The function then applied the query string to the events feature layer's layer definition. When applied, a request is made to the server and the server responds with the appropriate information for

the features that need to be added to the map. The events feature layer is then reset using the `Map` object's `setLayerDefintion` method.

5.2.4 Download Data Tool

The `downloadData` function is triggered by a listener on the Extract Data button that appears in the Download Data accordion pane. The function uses the current download data query string that is built through the query tools. If the user has not narrowed their search with the use of the query tools, a default query string is used.

Upon activation, the application sends a request to a geoprocessing service with the required parameters. The geoprocessing service was published from a tool made with the ArcGIS Model Builder (Figure 5.12). The tool is responsible for selecting records from a feature class, extracting the data into shapefile format, and delivering the shapefile in a zipped folder to the user as a downloadable file.

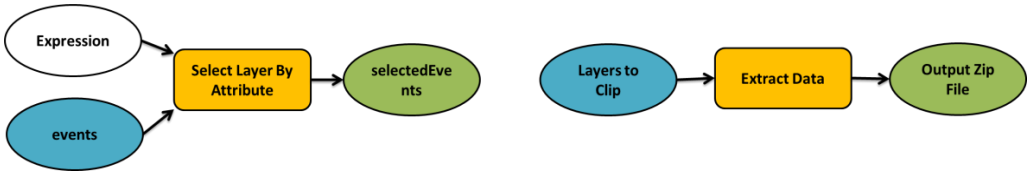


Figure 5-12: Workflow of Extract events tool

5.2.5 Submit Observations Form

When the user completes the Submit Observations form and pushes the Submit button from the user interface, a function that is responsible for submitting the observation event data is called. Within the function, a graphic object is created with the geometry specified in the latitude and longitude NumberTextBoxes (a Dijit) and the attributes specified by each of the other options in the form. The user ID entered upon the user's visit to the site is also saved as an attribute of the observation. Once the graphic is created, it is sent to the server using the ArcGIS API for JavaScript function `applyEdits` function that can only be called on feature services.

5.3 Summary

The web application consists of two tabs, Map and My Observations, intended for two different types of users. The tabs are composed of a map, an identify popup tool, and query tools. The My Observations tab also includes a Submit Observation form and a Download Data tool. These tools compose the functional requirements of the application and drive the dynamic components of the website.

Chapter 6 – Implementation of the Mobile Application

As stated, the mobile application is intended for users who are on the water and are interested in tracking their trips and submitting observations. The application was designed to allow users to visualize their trip and record survey events (tracks, observations) locally on the device in addition to syncing them with the server. This chapter discusses the implementation of the mobile application. Section 6.1 describes the User interface of each of the application's Views. Section 6.2 describes the functional components of each of the Views.

6.1 Mobile Application's User Interface

There are three Views associated with the mobile application: Home, Survey, and Observation. Each of these Views is associated with an Activity, which is essentially a custom class. From this point on, all future references to Activities will be in `UpperCamelCasing and` Views will be in `lowerCamelCasing.xml` followed by .xml. Upon launch of the application, the user is presented with the `home.xml` View (Figure 6-1). This View has the application's title, a settings button, and the "Start a Trip" button.

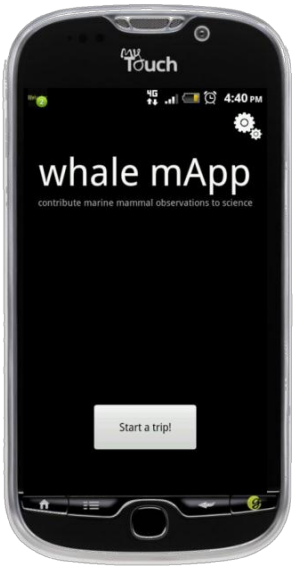


Figure 6-1: *Home* View of the mobile application

On selecting the settings menu, the user is allowed to only change the username that will be associated with the observations they submit. The list made available to the user to choose from is composed of "anonymous" along with each of the email accounts associated with their phone (Figure 6-2).

Figure 6-2: Settings Menu

The "Start a trip" button simply closes the `home.xml` View and brings the user to the Main Activity of the application: the `SurveyMap`. The `surveyMap.xml` view is composed of a map and a button that allows the user to record the observation (Figure 6-3). This View allows the user to visualize their trip with minimal effort. Their current location is represented by a blue dot. Every five minutes, the user's location is recorded as a position update and a new black dot appears on the screen. Each time a new observation is recorded, a symbol is added to the map to their marine mammal's category: whale, dolphin or porpoise, seal or sea lion, or unknown.

Figure 6-3: Survey Map View

When the user selects the button to record a new observation, they are presented with a form to fill out (Figure 6-4) in the `observation.xml` View of the `Observation` Activity. The user must select a species category before any other information can be added to the form. After selecting a category, the user can scroll through the remaining details to add. The required information is bolded and the unrequired information isn't.

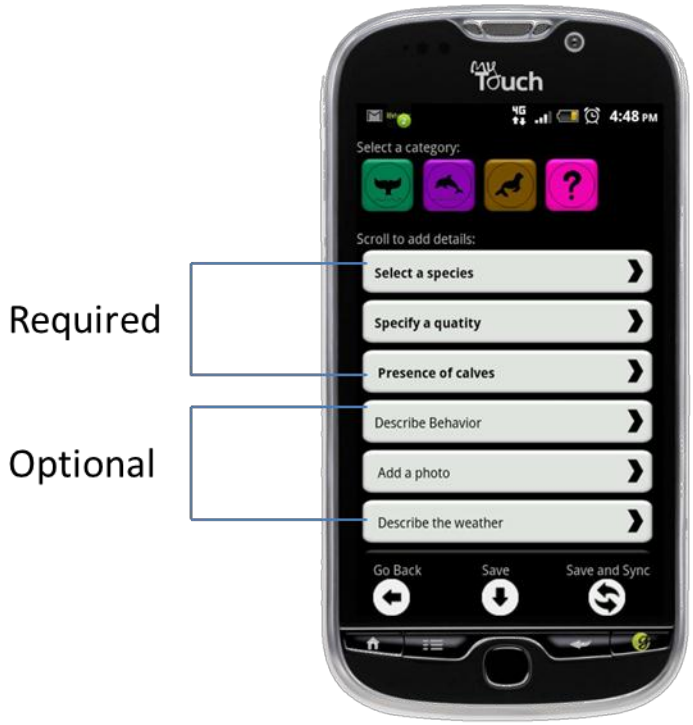
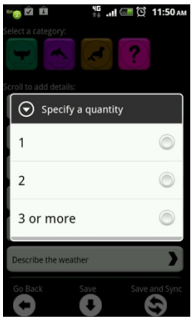


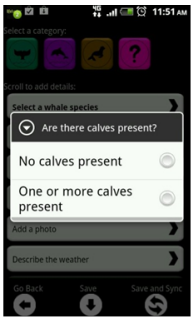
Figure 6-4: Observation View showing the required and optional components in the form

Each time the user selects one of the components, they are presented with a dialog box with options to choose from. Figure 6-5 illustrates each of the dialog boxes that are presented when the user selects the whale category.

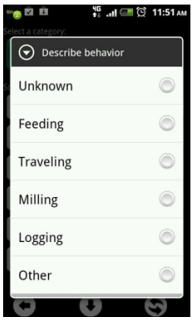
A) Select a whale species



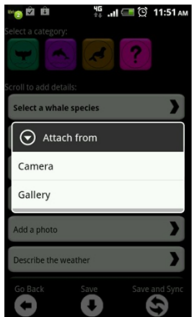
B) Select a quantity



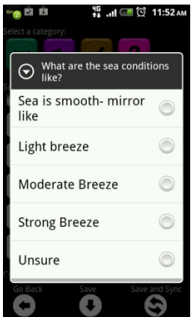
C) Presence of Calves



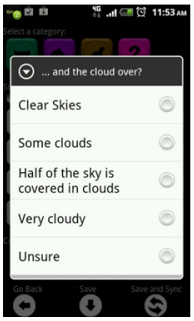
D) Describe Behavior



E) Add a photo



F) Describe the Beaufort



G) Describe the weather

Figure 6-5: Screenshots from the dialog boxes in the Observation form

At the bottom of the `observation.xml` View, the user has the option to "Go Back," "Save," and "Save and Sync." The "Go Back" closes the `Observation` Activity without saving. The "Save" button saves the observation locally on the phone, and the "Save and Sync" button saves the data locally on the phone in addition to syncing it with the server.

6.2 Functional Components of the Activities

This section describes the functional components of the mobile application by the *Activity* that they are associated with. Each of the Activitity's methods will be in `lowerCamelCasing()` followed by parenthesis and all objects appear in `lowerCamelCasing.`

Essential to understanding the functional components of the application is the Android Life Cycle (Activity, 2012). The life cycle describes the states and workflow of how applications are handled in devices running an Android operating system. Figure 6-6 illustrates the life cycle. Most of the functional components occur in the `onCreate()` function, but the `onResume()` and `onPause()` also play an important role in some cases.

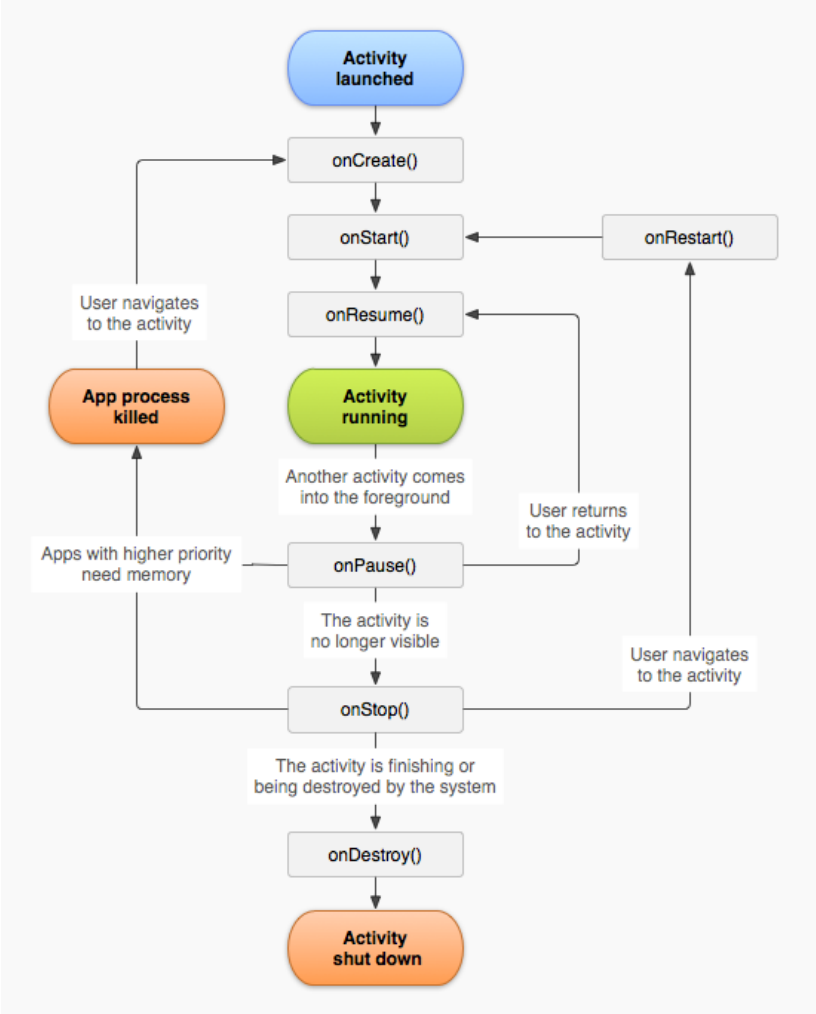


Figure 6-6: Android Life Cycle
Source: http://developer.android.com/reference/android/app/Activity.html

While the `Home` Activity is essential to the application, it contains no functional components. The two Views with functional components in their Activities are the `surveyMap.xml` and the `observation.xml` View. There is also a helper Activity with functional components that doesn't have a View: the `EventsDatabaseManager`. This Activity is responsible for handling the data stored locally on the device.

6.2.1 Survey Map Activity

The `SurveyMap` Activity has several functional components which include: the `map` object, `positionUpdates()` method and handler, `onStatusChange` Listener, and the `updateEventsLayer()` Method (Figure 6-7).

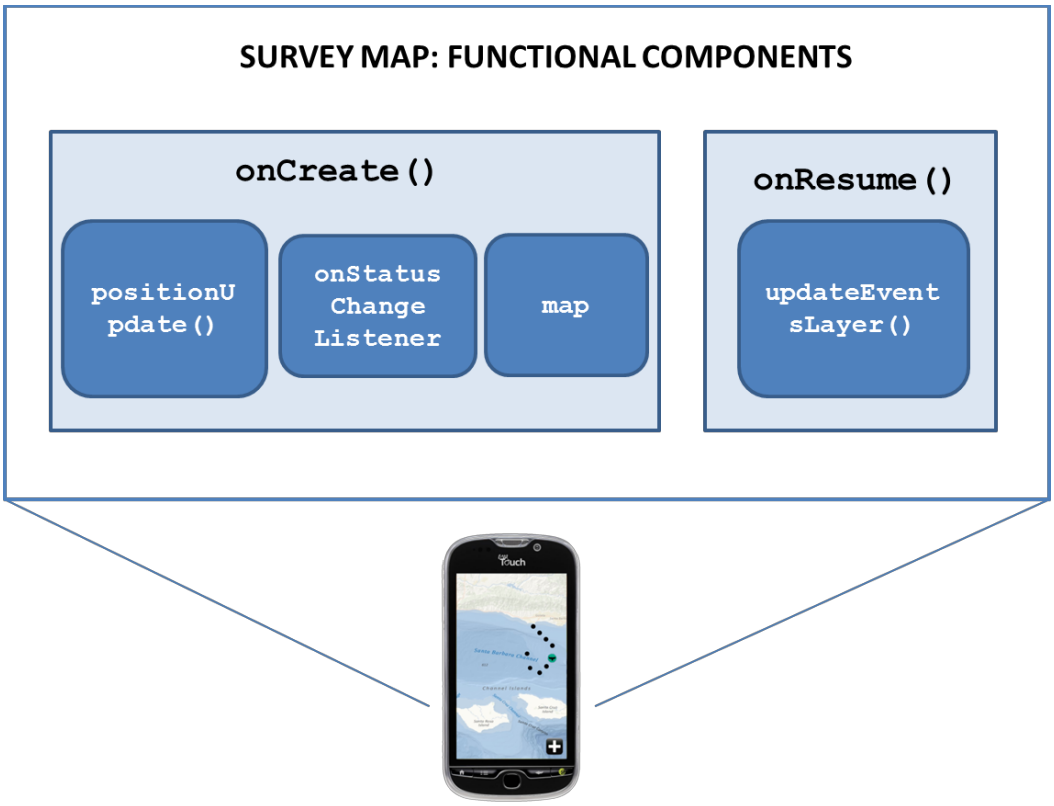


Figure 6-7: Functional components of the Survey Map Activity

When the Activity is first called by the`home.xml` View, the `onCreate()` function is called. The `onCreate()` function is an essential part of the Android Life Cycle. The `map` object is initialized and Esri's Ocean basemap layer is added to it. Additionally in the `onCreate()` function, the `onStatusChange` listener is declared. This status change listener handles everything related to the phone's position. It uses the Android location service (Location & Maps, 2012), which triggers a function when the position has changed. The listener then gets the latitude and longitude from the location service. The latitude and longitude are used to change the position of the location marker on the map. The envelope of the map is then adjusted to extend three miles around the new position.

After the `onCreate()` method is complete, the `onResume()` function is called. Within this function, the `updateEventsLayer()` Method is called. This method uses a static class to create a collection of features with geometry and attributes of each record stored in the local database. It then creates graphics from each of the features in the collection and adds it to the map. Also in the `onResume()` function, a repeating task is started to automatically record the user's position and draw a black dot on the map every five minutes. This repeating task utilizes Android's `Handler` class which repeats a particular action with a delayed time increment.

When the user pushes the button to add an observation, the `onPause()` function is called for the `SurveyMap` Activity. In this function, the map is paused and the repeating task is stopped so that the position won't be updated during a recording of an observation.

6.2.2 Observation Activity

When the `Observation` Activity is started, the `onCreate()` function is called. This Activity is primarily responsible for responding to user actions as they fill out the form, saving, and syncing the data (Figure 6-8).

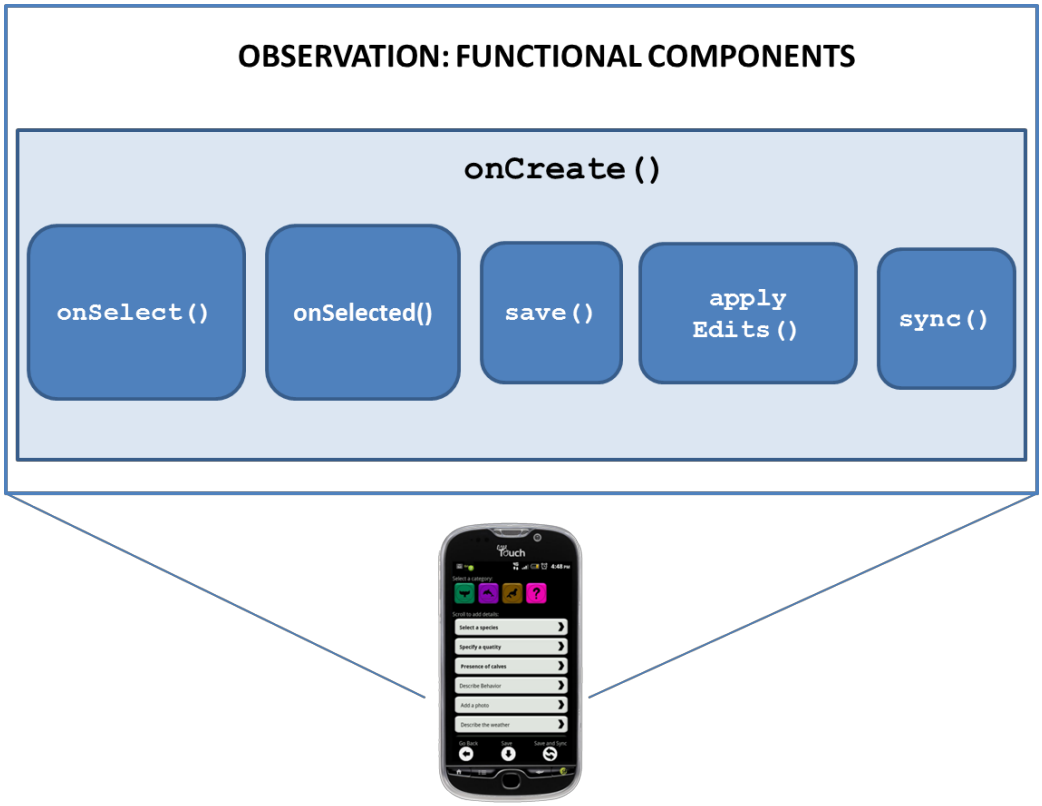


Figure 6-8: Functional components of the Observation Activity

As previously mentioned, the first component of information required is the marine species category. When the user selects a category, the first of this Activity's many `onSelected()` functions are called. The Species category's `onSelect()` method adjusts the appearance of the detail components required, depending on which of the categories the user selected: whale, dolphin or porpoise, seal or sea lion, unknown (Figure 6-9).

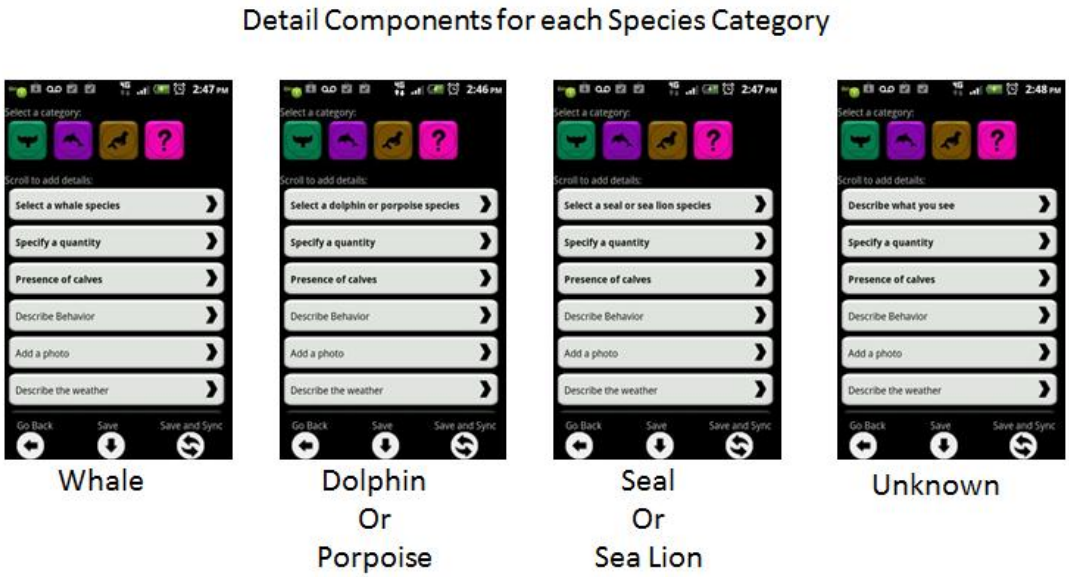


Figure 6-9: Detail components provided to the user upon species category selection

After the initial selection, each of the detail components have a similar workflow. On selection, the component's `onSelect()` method is called. These methods build a list of options for the user to select from and populate them in an alert dialog box that is presented to the user (AlertDialog, 2012). The AlertDialog box has a listener associated with it that waits for the user to tap a selection. Once the selection is tapped, the corresponding `onSelected()` method is called. This method is responsible for updating the component detail's description with the choice the user made. This serves as a visual confirmation of what the user is recording (Figure 6-10).

Figure 6-10: Comparison of beginning and ending of the observation form

The third functional component of the `Observation` Activity is the Save function. This function is called when either the "Save" or the "Sync and Save" button is selected. The first step the method takes is verifying that each of the required pieces of information has been recorded. Afterwards, each of the record's attributes is stored in the local database using methods defined in the `EventDatabaseManager` Activity.

The final component of the `Observation` Activity is the Sync function. This function is called whenever the "Sync and Save" button is pushed. The first step the function takes is to call the save method previously mentioned. This ensures that the observation is stored locally. The function then creates a graphic with a geometry passed to the `Observation` Activity from the `SurveyMap` Activity. The graphic is also assigned the attributes corresponding to the observation details. The graphic is then passed in a server REST API `applyEdits()` method to the server.

6.2.3 Events Database Manager Activity

The `EventsDatabaseManager` Activity is responsible for interfacing between the other activities and the database in which the events are stored (Ehrenstein, 2011). The functional components of this Activity are illustrated in Figure 6-11. In the `onCreate()` method, a new table is created that matches the events feature class schema if one doesn't exist. This should only occur on the initial installation of the application. The `open()` and `close()` methods can be called by other activities and are responsible for opening and closing the table so that actions can be performed on it.

Both of these methods are called by both the `SurveyMap` and the `Observation` Activities.

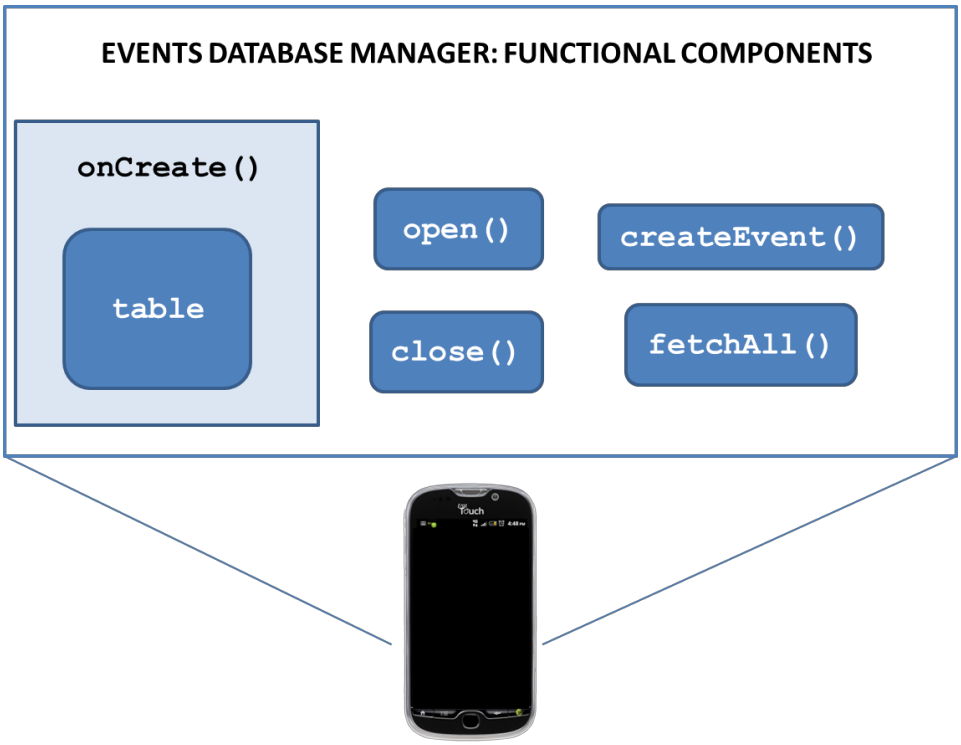


Figure 6-11: Functional components of the Events Database Manager Activity

The `fetchAll()` method is responsible for querying the table and returning all attributes for every record in the database. This method is called by the `SurveyMap` Activity. The `createEvents()` method creates a `contentValue` object and fills it with the attributes sent to it by the Activity that called the function. It then creates a new record in the database and populates it with the incoming attributes.

6.3 Summary

The mobile application consists of three different Views: `home.xml`, `surveyMap.xml`, and `observation.xml`. Each View has a corresponding Activity, and there is an additional Activity for interfacing with the database stored locally on the device. Each Activity has several functional components that are essential to the application.

Both the mobile and web applications play a fundamental role in the project solution. As previously mentioned, the web application allows both users who are actively collecting observations and those that are inactive and simply interested in exploring the data. Active users have access to advanced query tools, an observation submission form, and a data download tool. This mobile application provides a connectivity-hybrid solution for connected editing with data stored locally for future development of a disconnected editing solution. It complements the web application in that it provides a tool for allowing users to submit observations directly from the field.

Chapter 7 – Implications of Volunteered Geographic Information Software Development

With the web and mobile applications designed and developed, their implications should be considered. This chapter revisits some of the topics discussed in Chapter Two to understand the effects and implications of the system developed for this project, with an emphasis on the mobile application. Section 7.1 discuses techniques for ensuring enough quality data for scientific research using a VGI approach. Section 7.2 describes how spatiotemporal analysis can be used to identify patterns in VGI data and the differences between this project's data model and the ArcMarine model. Section 7.3 is dedicated to the connectivity issues with the current workflows and toolsets for mobile development. Lastly, Section 7.4 discusses the mobile application user interface and ideas for making it more engaging. The chapter closes with concluding remarks in Section 7.5.

7.1 Considerations for Volunteered Geographic Information and Science

To use VGI data in research, the right questions need to be asked of the right people; and enough people have to be asked those questions. The importance of this was discovered during a field test of an early version of the mobile application. Applications that are intended for use by the general public need to frame questions in ways that are easy to understand. The application was originally designed to ask the user how many calves were present with a group of mammals. Counting the number of animals present is a simple task when a group of whales is being surveyed. This is because whales tend to travel in groups of 1-4. However, determining the number of calves present in a group of dolphins is much more difficult because they travel in larger groups of 10-500. Rather than asking the user how many calves were present, the application was changed to simply ask whether there were calves present or not. The true/false response of calf presence can still be used to understand mammal behavior without an exact number.

This was an example of an easy change, which framed the question for a member of the general public. There are some cases in research when the questions cannot be simplified, and therefore should be reserved for trained users. In these cases, user accounts with varying permission levels can be created. For example, an expert user could have access to more involved questions whereas a public user could have access to general questions.

During this project, the application was tested on a single mobile device. In order to collect enough data for scientific research, the application would have to be deployed on a much larger scale (i.e.- Google Play Application Store). In doing so, the design of the system infrastructure would need to be reevaluated. Chapter Two discussed some citizen science-based web applications deployed using open source technologies. Unfortunately, it is difficult for small organizations to "set up a web server, learn open source (free) web-development Content Management Systems, and ultimately, create, design, and maintain their own website to support their own citizen science program needs," (Newman, Graham, Crall, & Laituri, 2011, p. 1853).

Esri has recently developed solutions for multi-scale deployment, as an alternate to open source technology. One solution is ArcGIS Online for Organizations, which gives organizations a mapping platform for sharing maps and editing through feature services. Another solution is ArcGIS Server on Amazon Cloud. Both of these solutions are flexible in terms of the number of users with access. They also do not require personnel to maintain the physical servers that all of the data lives on. Unfortunately, each of the Esri alternatives is expensive. The client would need to apply for an Esri Conservation grant to utilize the technology.

7.2 The Use of VGI in Marine Research

Chapter two discussed methods that are currently used in the field of marine research for storing and analyzing data. ArcMarine was described as a data model for storing a wide range of data in an efficient manner for analysis. However, its design demonstrated that it was not intended for volunteered geographic information. The data model would need to be modified before it could be used for the project by eliminating most of the tables, and adding new tables and necessary attributes. The conceptual database designed for this project was similar to ArcMarine in that it associated non-spatial elements (such as an observer or survey) to spatial elements (such as an event). In this respect, the survey table of this project's conceptual model was similar to the survey info table in the ArcMarine data model. Similarly, this project's events feature class is similar to the ArcMarine location series feature class.

Regardless of the similarities in the feature classes and tables, the models diverged during the design of the logical model. Because of the nature of Web and Mobile GIS projects, the conceptual model was consolidated into a flat file structure to improve the speed of queries performed on the database. This made ArcMarine an unrealistic data model for the project's scope.

A study that performed spatio-temporal analysis on Flickr photos found that social media data in conjunction with geovisualization methods could be used to understand mobility and social dynamics in urban systems (SAGL, Resch, Hawelka, & Beinat, 2012). Similar analysis could be performed on marine mammal observations contributed using a VGI approach. Without taking this into account, improper assumptions and assessments could be made during analysis. Spatio-temporal techniques, including those in ArcGIS tool suite can help understand these space-time clusters. For example, the Grouping Analysis tool can group features based on attributes and with optional spatial/temporal constraints. Similarly, the Cluster & Outlier Tool uses Local Moran's I statistic to identify statistically significant hot spots, cold spots, and spatial outliers. Tools like this could help determine if the same animal is likely being reported several times by different users. They could also be used to identify groups of events that belong to a particular trip and user.

7.3 Connectivity in VGI Web and Mobile GIS Applications

There were a couple of significant obstacles faced during the course of the project, both of which pertain to connectivity of the mobile application. Providing a disconnected editing experience using the current ArcGIS Runtime Software Development Kit (SDK) for Android was challenging. This is because the kit does not include the necessary tools

for seamless disconnected editing. A work around was created for storing data in a local SQLLite database on the phone. However, the application does not attempt to submit observation data more than once. This is not ideal as users do not always have connectivity. This also forces the user to submit observations as they record them rather than retroactively at their leisure.

The second issue addressed related to connectivity pertains to the basemaps used in the mobile application. In a true disconnected environment, the user would be accessing basemaps stored locally on their device. The geographic scope was limited for the project to an area that had connectivity. Because of this, Esri online basemaps were utilized. If the application was designed for truly disconnected editing, cached basemaps for the user's location would have to be delivered to the mobile device. If the user requested cached maps through the mobile application, the application could be redesigned to identify the user's location, and download the maps necessary for their region. Because of the nature of cached maps, storing them on the Android device could monopolize a significant amount of memory on the phone. For example, the project's study area requires approximately 4 GB of memory in cached maps.

7.4 Mobile User Interface Critique

The four areas of proper user interface design discussed in Chapter Two were simplify the task, reduce memorization, plan for error, and test the usability of the application. The task of submitting observations was simplified in the project by taking the scientific form used in the client's research and turning it into an easily accessible form in either a mobile and web application. Reducing memorization was handled by providing the user with a list of options to choose from rather than asking them to manually enter the requested information. A feedback system was designed in the mobile application in planning for error. After completing each element in the observation form, the user is presented with a visual confirmation of their selection. This feedback system, allows them to change their selection if desired.

The usability of the mobile and web applications were not tested over the course of this project. There are several components of the user interface that could be improved to make the system more engaging. One approach to this is to game-ify the experience. Adding game elements to the applications could provide the users with opportunities for collaboration and a feeling of belonging to something greater than themselves (McGonigal, 2011). Some game elements include feedback loops that could tell the user information such as how many observations they've submitted to the community average. Another game element for engaging users is to challenge them intellectually. Users could be challenged by competing or working as a team to collect data and answer trivia questions. This method also encourages collaboration among the application users. If users don't already have a team to collaborate with, a notification system could be used to inform users if they are in close proximity of other users.

7.5 Summary

In conclusion, Section 7.1 demonstrated how the right questions need to be asked to the right people. It also discussed how expanding VGI projects to collect enough data for research is either time consuming or expensive. Section 7.3 explored the differences

between the data model developed for this project and the existing ArcMarine model. It also described how GIS can be used to identify spatiotemporal patterns in data. In Section 7.3 the disadvantages surrounding connectivity and Mobile GIS. A better toolset is needed for disconnected editing with the Runtime SDK for Android. Additionally, better workflows are needed for handling offline basemaps. Finally, Section 7.4 discussed the design of the mobile user interface and how error was accounted for, tasks were simplified, and memorization was reduced. Testing of the mobile application usability still needs to be performed, and the user interface could be improved using game elements to make it more engaging.

Chapter 8 – Conclusions and Future Work

The challenge that the client faced was how to generate a high volume of quality data for her long-term study on marine mammal migratory behavior and human impact. In order to reduce the cost and effort required in the data collection process, Stelle decided to explore a VGI approach to collecting this data, which incorporates the collection of data by both researchers and members of the general public. The proposed solution consisted of web and mobile applications and a geodatabase for storing observations collected by researchers and members of the general public.

A conceptual model was used to describe the client's business model and logical models for both the server and client side were designed. The server side logical model consisted of a single feature class for holding all submitted events and an associated attachment table to hold all evidence, such as photos. The client side logical model consisted of a single table in a SQLLite database on the mobile device. All evidence was stored as a path to its location in the Photo Gallery on the Android device. Domains and subtypes were incorporated into the server side model to ensure data integrity.

The web application was designed for the user to visualize all data on a map and query the data based on the type of species, date range, and event type. It was also designed to allow users to download selected data in a shapefile format. The web application interface includes a form for submitting observations upon login to the system.

The mobile application was designed to allow users to visualize their whale watching trip and corresponding recorded observations. This is done by rendering a new position update on a map every five minutes. A form was created that allows users to submit new observations and corresponding evidence. The user can choose to associate their observations with a user account or submit them anonymously. The data associated with the observations and position updates are stored locally on the device and synced with the server when submitted. Upon which, a new symbol is rendered on the map for their observation.

The purpose of this project was to demonstrate the capability of the system with proofs of concept. During development of the proofs of concept, some recommendations were determined pertaining to the user experience and expanding the project to a larger audience.

The first suggestion for improving the user experience is to give users more control over their data. On the mobile application, the user should be able to retroactively submit observations after a trip. Currently, the user has a single chance to submit the observation. In some cases, the user may also desire to edit their observations. For example, they may have made a mistake in the details of an observation or forgot to submit a photo with the observation. Editing capabilities of observations should be added to the web application interface. This capability would, for example, allows users to add photos to previously submitted observations.

There are other suggestions for improving the user interface and making the applications more engaging. The applications could include game elements such as feedback loops, collaboration, and competition to give the user a feeling of belonging to something greater than themselves. Feedback loops could tell the user how many

observations they've submitted compared to the others in the community. Specific examples of how these elements could be incorporated are discussed in Chapter Seven.

In order for the application to be deployed to a wide audience, several considerations need to be taken into account. Currently, the applications are only designed to allow users to select from animals in southern California. The available species list would need to be expanded to a much larger scale. With a larger geographic scope, the mobile application would need to be capable of working in completely disconnected environments. To do so, basemaps would need to be cached and available for download by users in these environments.

As the above suggestions are incorporated, it is essential that the applications are tested on small user groups before deploying at a large scale. Testing will provide important feedback on the user experience. These suggestions for future work will produce a system that is ready for use by a wide range of users.

In summary, this project demonstrated how volunteered geographic information could be used alongside Web and Mobile GIS to manage marine mammal observations. Several suggestions for improvement have been identified for further development. Upon deployment, the data collected using the applications can play a key role in understanding human impact on, and behaviors of, migrating marine mammals.

Works Cited

Activity. (2012, 10 12). Retrieved 10 14, 2012, from Android Developer: http://developer.android.com/reference/android/app/Activity.html

AlertDialog. (2012, 10 12). Retrieved 10 14, 2012, from Android Developer: http://developer.android.com/reference/android/app/AlertDialog.html

Android 2.3.3 APIs. (2012). Retrieved 2012, from Android Developer : http://developer.android.com/about/versions/android-2.3.3.html

Android Developer Reference. (2012). Retrieved 10 9, 2012, from Intent: http://developer.android.com/reference/android/content/Intent.html

Aberley, D., & Sieber, R. (2002). *PPGIS.net Open Forum on Participatory Geographic Information Systems and Technologies*. Retrieved 12 2012, from About PPGIS: http://www.ppgis.net/ppgis.htm

Aguilar, M. (2012, July). *Gizmodo.com*. Retrieved July 2012, from Only 10-Percent of Androids Have Ice Cream Sandwich After Eight Months: http://gizmodo.com/5923143/only-10-percent-of-androids-have-ice-cream-sandwich-after-eight-months

Apple. (1987). *Human Interface Guidelines: The Apple Desktop Interface*. Addison-Wesley Publishing Company Inc: New York.

Bauer, J. R. (2012, 6). *Assessing the Robustness of Web Feature Services Necessary to Satisfy the Requirements of Coastal Management Applications*. *MA Thesis*. Retrieved 10 2012, from Oregon State University Library: http://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/30062/BauerJennifer2012.pdf?sequence=1

Best, B., Halpin, P., Fujioka, E., Read, A., Qian, S. S., Hazen, L., et al. (2006). Geospatial Web Services Within a Scientific Workflow: Predicting Marine Mammal Habitats in a Dynamic Environment. *Ecological Informatics*.

Connors, J. P., Lei, S., & Kelly, M. (2011). Citizen Science in the Age of Neogeography: Utilizing Volunteered Geographic Information for Environmental Monitoring. *Annals of the Association of American Geographers*, Forthcoming 2012.

Davidson, A. D., Boyer, A. G., Kim, H., Pompa-Mansilla, S., Hamilton, M. J., Costa, D. P., et al. (2012). Drivers and Hotspots of Extinction Risk in Marine Mammals. *Proceedings of the National Academy of Sciences of the United States of America*, 3395-3400.

Dojo. (2011). *Dojo toolkit*. Retrieved 10 1, 2012, from Dijit Documentation for Version 1.6: http://dojotoolkit.org/reference-guide/1.6/dijit/index.html#dijit-index

Ehrenstein, C. (2011). Android App Development & Design. *Learn By Video*. Peach Pit Press.

Esri. (2011, Dec 14). *Android Runtime SDK for Android*. Retrieved Feb 22, 2012, from Esri: http://resources.arcgis.com/content/arcgis-android/sdk

Esri. (2012, Feb 1). *What is a Feature Service?* Retrieved Feb 20, 2012, from Esri: http://help.arcgis.com/en/arcgisserver/10.0/help/arcgis_server_dotnet_help/index.html#//009300000020000000

Esri. (n.d.). *ArcGIS Server Training Seminars*. Retrieved Feb 20, 2012, from Esri: http://training.esri.com/campus/catalog/ServerSeminars.cfm

Foley, M. M., Halpern, B. S., & Micheli, F. (2010). Guiding Ecological Principles for Marine Spatial Planning. *Marine Policy*, 955-966.

Francica, J., & Schutzberg, A. (2011, Sep 13). Top Skills Needed to be Successful in a GIS Career. *Directions Magazine Podcast*. Podcast retrieved from iTunes.

Fu, P., & Sun, J. (2011). *Web GIS Principles & Applications*. Redlands: Esri Press.
Geospatial Innovation Facility. (2012). *About OakMapper*. Retrieved from OakMapper Sudden Oak Death: http://www.oakmapper.org
Glennon, A. (2011). *Geyser Notebook*. Retrieved February 14, 2012, from Android Market: https://market.android.com/details?id=net.geysers.app.geysernotebook
Glennon, A. (2012, February 15). personal communication. (M. King, Interviewer)
Goodchild, M. F. (2007). Citizens as Sensors: the World of Volunteered Geography. *GeoJournal*, 211-221.
Google. (2012). *Android*. Retrieved Feb 23, 2012, from Android Developers: http://developer.android.com/index.html
Halpern, B. (2008). A Global Map of Human Impact on Marine Ecosystems. *Science*, 948-952.
Halpin, P. (2004). *NOAA Office of Science & Technology / EcoGIS: GIS Tools for Ecosystem Approaches to Management*. Retrieved August 15, 2012, from Spatial Analysis Needs for Marine Ecosystem Management: Habitat Characterization, Spatio-Temporal Models and Connectivity Analysis Frameworks: http://www.st.nmfs.noaa.gov/ecogis/Workshop2004/Documents/Halpin_presentation.pdf
Jones, J. S., & Ganey, S. (2009). Building the Legal and Institutional Framework. *Ecosystem-based Management for the Oceans*, 162-179.
Location & Maps. (2012, 10 12). Retrieved 10 14, 2012, from Android Developer: http://developer.android.com/guide/topics/location/index.html
Lloyd, C. (2012, September 20). *Android Community*. Retrieved October 1, 2012, from Verizon sees more Android activations than all other carriers: http://androidcommunity.com/verizon-sees-more-android-activations-than-all-other-carriers-20120920/
Mandel, T. (1997). *The Elements of User Interface Design*. New York: John Wiley & Sons.
McGonigal, J. (2011). *Reality is Broken: Why Games Make Us Better and How They Can Change the World*. New York: The Penguin Press.
National Ocean Service. (2011, 1 6). *Marine Mammal Sightings Database*. Retrieved 10 16, 2012, from Channel Islands National Marine Sanctuary: http://www.cisanctuary.org/mammals/
Newman, G., Graham, J., Crall, A., & Laituri, M. (2011). The art and science of multi-scale citizen science support. *Ecological Informatics*, 218-226.
Newman, G., Zimmerman, D., & Crall, A. (2010). User-friendly web mapping: lessons from a citizen science website. *International Journal of Geographic Information Science*, 1851-1869.
NOAA. (2011, 7 31). *NOAA Coastwatch: West Coast Regional Node*. Retrieved 10 16, 2012, from Coasta watch Browser: http://coastwatch.pfeg.noaa.gov/coastwatch/CWBrowser.jsp
Norman, D. A. (1988). *The Design of Everyday Things*. New York: DoubleDay.
O'Malley, R. (2010, 9 15). *Ocean Productivity*. Retrieved 10 16, 2012, from http://www.science.oregonstate.edu/ocean.productivity/
Pelton, M. R., & Manen, F. T. (1996). Benefits and Pitfalls of Long-Term Research: A Case Study of Black Bears in Great Smoky Mountain National Park. *Wildlife Society Bulletin*, 443-450.
Pian, S., & Menier, D. (2011). The use of a geodatabase to carry out a multivariate analysis of coastline variations at various time and spave scales. *Journal of Coast Research Special Issue 64*, 1722-1726.

Pompa, S., Ehrlich, P. R., & Ceballos, G. (2011). Global Distributions and Conservation of Marine Mammals. *Proceedings of the National Academy of Sciences of the United States of America*, 13600-13605.

Ruckelshaus, M., Klinger, T., Knowlton, N., & DeMaster, D. P. (2008). Marine Ecosystem-based Management in Practice: Scientific and Governance Challenges. *BioScience*, 53-63.

SAGL, G., Resch, B., Hawelka, B., & Beinat, E. (2012, 3 7). *From Social Sensor Data to Collective Human Behaviour Patterns - Analysing & Visualising Spatio-Temporal Dynamics in Urban Environments*. Retrieved 10 23, 2012, from GIS Point: http://gispoint.de/index.php?id=5&tx_browser_pi1%5BnewsUid%5D=682&cHash=208290d911

Steiniger, S., Neun, M., & Alistair, E. (n.d.). *University of Minnestota - Spatial Database and Spatial Data Mining Research Group*. Retrieved Feb 23, 2012, from Foundations of Location Based Services: http://www.spatial.cs.umn.edu/Courses/Spring10/8715/papers/IM7_steiniger.pdf

Stelle, L. (2008). Activity budget and diving behavior of grey whales (Eschrichtius rubustus) in feeding grounds off coastal British Columbia. *Marine Mammal Science Vol 24. No 3*, 462-478.

Stelle, L. (2012, Feb 10). Data Integration: Marine Mammals and Their Ecosystems Citizen Science "Whale mAPP" [Powerpoint slides]. Redlands, California: Unpublished.

The Cornell Lab of Ornithology. (2011). *eBird*. Retrieved Jan 31, 2012, from eBird: http://ebird.org

Timoney, B. (2012, 8 1). *How the Public Actually Uses Local Government Web Maps: Metrics from Denver*. Retrieved 10 1, 2012, from Map Brief: http://mapbrief.com/2012/08/01/how-the-public-actually-uses-local-government-web-maps-metrics-from-denver/

Tsou, M.-H. (2004). Integrating Mobile GIS and Wireless Internet Map Servers for Environmental Monitoring and Management. *Cartography and Geographic Information Science*, 153-165.

University of Washington Sea Grant Institude. (2011, 8 11). *Impacts and Outcomes of Mature Coastal Web Atlases: California*. Retrieved 10 2012, from YouTube: http://www.youtube.com/watch?v=cLPq65Rj0uA

UW Center for Game Science. (2011). *Fold it: Solve Puzzles for Science*. Retrieved Jan 31, 2012, from Homepage: http://fold.it/portal/

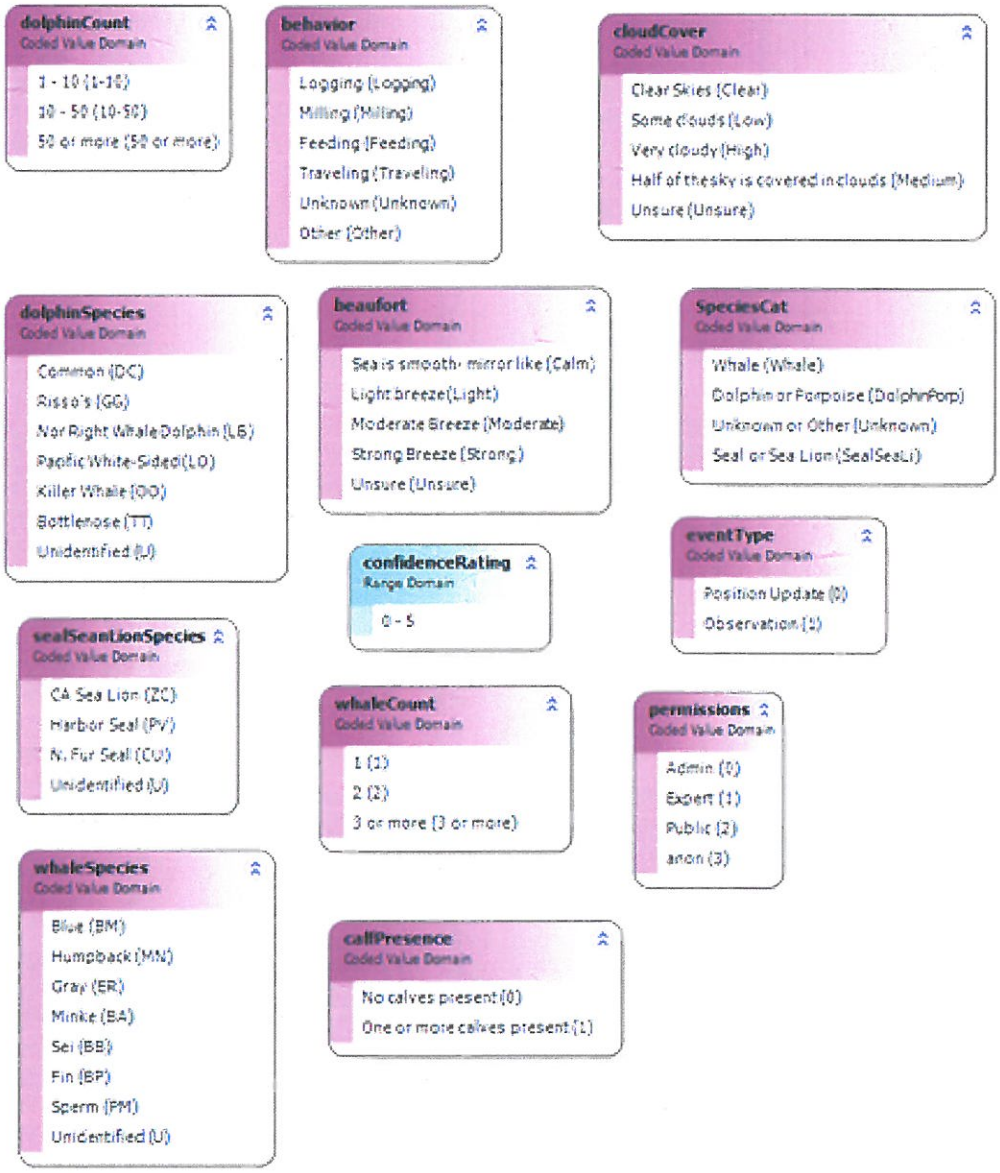
Viswanathan, P. (2012). *About.com*. Retrieved October 1, 2012, from Android OS Vs. Apple iOS – Which is Better for Developers?: http://mobiledevices.about.com/od/kindattentiondevelopers/tp/Android-Os-Vs-Apple-Ios-Which-Is-Better-For-Developers.htm

Wagtendonk, A. J., & De Jeu, R. A. (2007). Sensible Field Computing: Evaluating the Use of Mobile GIS Methods in Scientific Fieldwork. *Photogrammetric Engineering & Remote Sensing Vol. 73, No. 6*, 651-662.

Walker, D. A., Halfpenny, J. C., Walker, M. D., & Wessman, C. (1993). Long-term Studies of Snow-Vegetation Interactions. *BioScience*, 287-301.

Wright, D., Blongewicz, M., Halpin, P. N., & Breman, J. (2007). *Arc Marine*. Redlands: ESRI Press.

Appendix A. Geodatabase Domains



Appendix B. Mobile Application Activities Code

```
 1 package edu.gis.spatial.redlands.edu.Cohort21.melodi_king;
 2
 3 import java.io.File;
65
66 public class Observation extends Activity {
67     int SELECT_PICTURE = 1;
68     public double locy;
69     public double locx;
70     public String userID;
71     int i = 999;
72     public boolean catPicked = false;
73     // private Spinner speciesType;
74
75     // database object
76     private EventsDBManager mDbManager;
77
78     // initial/default variable values
79     private String category = "Unknown or Other";
80     private String speciesType = "Unidentified";
81     private String count = "No calves present";
82     private String notes = "";
83     private String behavior = "Unknown";
84     private String calves = "No calves present";
85     public String seaString = "";
86     public String weatherString = "";
87     private Cursor mCursor;
88     private RatingBar ratingBar;
89     String txtRating;
90     public boolean speciesPicked = false;
91     public boolean countPicked = false;
92     public boolean calvesPicked = false;
93     public int catInt;
94     private int rateInt = 2;
95
96     // no longer using credentials, can delete
97     // UserCredentials credentials = new UserCredentials();
98     // ArcGISFeatureLayer fLayer = new
99     //
   ArcGISFeatureLayer("http://gis.spatial.redlands.edu/ArcGIS/rest/services/melodi_king/
   events/FeatureServer/1",
100    // MODE.SNAPSHOT, UserCredentials(credentials));
101    ArcGISFeatureLayer fLayer = new ArcGISFeatureLayer(
102            "http://gis.spatial.redlands.edu/ArcGIS/rest/services/melodi_king/events/
   FeatureServer/1",
103            MODE.SNAPSHOT);
104
105    int permissions = 2;
106    String layerDefinition = "{\"currentVersion\":10.02,\"id\":1,\"name\":\"events
   \",\"type\":\"Feature Layer\",\"displayField\":\"ObserverID\",\"description\":\"\",
   \"copyrightText\":\"\",\"relationships\":[],\"geometryType\":\"esriGeometryPoint\",
   \"minScale\":0,\"maxScale\":0,\"extent\":{\"xmin\":-14115658.1665,\"ymin
   \":3240128.2558,\"xmax\":-13061473.3222,\"ymax\":4321833.7614,\"spatialReference\":
   {\"wkid\":102113}},\"drawingInfo\":{\"renderer\":{\"type\":\"uniqueValue\",
   \"field1\":\"SpeciesCategory\",\"field2\" : null,\"field3\" : null,\"fieldDelimiter
```

\":\", \",\"defaultSymbol\":{\"type\":\"esriSMS\",\"style\":\"esriSMSCircle\",
\"color\":[0,158,37,255],\"size\":4,\"angle\":0,\"xoffset\":0,\"yoffset\":0,
\"outline\":{\"color\":[0,0,0,255],\"width\":1}},\"defaultLabel\":\"\u003call other
values\u003e\",\"uniqueValueInfos\":[{\"value\":\"2\",\"label\":\"Dolphin or
Porpoise\",\"description\":\"\",\"symbol\":{\"type\":\"esriPMS\",\"url\":
\"B698AF73\",\"imageData\":\"iVBORw0KGgoAAAANSUhEUgAAABgAAAAZCAYAAAArK
+5dAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADsQBlSsOGwAAAwFJREFUSInllWtIk2EUx397t/mmstSZS
20i3iorDbbuYhQlkUFGYGFEd6w+FEVpRZFhSR
+EUAlBsQhaQh/EriSRYjUUwyS3LphamJdZ5BtUZi/u0pcarc1pQRB0Pj1wzvP/PecznlU/GVT/RcAtZE1ux
NISQsiJFSFSpAZ/tSNtbeJmnbABNj/CJBN/towUVe4UF496zEN6rksJZQIt38bBc01lC5rFE0Vsiw3/xZgl3
j2mF6edSxZnq/po4ty9lHITQ9ACFMW7+D0giR5XtB59oYDtyYEyOPiOlGeXJDMfBHAzDUASjjIevbgYBSAqe
hJxKhMJyv7K1+o4lA38GIcgEsxQnmFkVUiwHsGqOUcABJdVHHYI1pLLPspI4OcrA6ahh5Ss9cvoFC8kR8kz9
E5sNNCHSaKfSXpNokeTpFFEXcCVrJ1ZldUU6rNZrOMCfggS1uSMHCPai5wxEtQSyKbOcx0jIShQ2aEFuqopp
g8KhZim5wJfgCBTIoRUFJLiZd4NvlkspMgND
+VQI2RFVRwgGc0BxpYob9Nx9glUqAMduJEwuYFSCDFQ/yHWTBzlCuEEEEwYcE/
+3w12fGeAWEG6Wwij1hmEkAgfbykHhORxBNFnDvaiZPpGJhKDACt3PMYOi+Ag9GhYT5GHqXK47WxJLOdMzh
+GVoBwS3uxMUQfYN
+ARFafYNLYpOvUihQoELtnfR366ez36Ktq0HyA4iWkk484s5GPUnKH6IALlzus51RnyArD9okSXriN4Mcpr0
u0ZovWSXzTjUiGrToSaSdRvTMYArRWHlIFPHoiKGeqxhYzjt637TTePxXPZ
+rYraUltvE7blveTVPSyQdtJFAKiN8wYqZSOKwY8eKmcWsoY36z328LHpCg3VCgAwUzkqeLVIimLp5uiGFJY
KGUDSEo/veUIBworFwv3OQziPXKav1pTXmNs1ltgPIqaTtdAetpeBMDSBQC4Lgwv5xmOEeSRy8e1k
+cRL4OpbOuB9OLobnQIZPpzze7X/ky/zPAd8AJlX1ZOTUiUwAAAAASUVORK5CYII=\",\"contentType\":
\"image/png\",\"color\" : null,\"width\":18,\"height\":18.75,\"angle\":0,\"xoffset
\":0,\"yoffset\":0}},{\"value\":\"0\",\"label\":\"None\",\"description\":\"\",
\"symbol\":{\"type\":\"esriPMS\",\"url\":\"C466E8E2\",\"imageData\":
\"iVBORw0KGgoAAAANSUhEUgAAAAwAAAAMCAYAAABWdVznAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADs
QBlSsOGwAAAJ1JREFUKJGt0bEKQVEYB/DfQDezDBZ5C6U8AQm7V7CIyWSUhzGZWL2EzAaZ9WWxUPderlLOeM
75nf7/75T8uEr/AHWs0Hue7zDH8ROo4oBmam
+ADlo4ZUCSJIuISF9OP7TGMAMiYvQlfheVfKT6F1BGLQ/OaBSAOy75DpuImBSALW75Dkv0ZacEV0x5H
+sVbdl/2GOG0yfw6jEuiOUB/Z0hYNqEvhsAAAAASUVORK5CYII=\",\"contentType\":\"image/png\",
\"color\" : null,\"width\":9,\"height\":9,\"angle\":0,\"xoffset\":0,\"yoffset\":0}},
{\"value\":\"3\",\"label\":\"Seal or Sea Lion\",\"description\":\"\",\"symbol\":
{\"type\":\"esriPMS\",\"url\":\"82D47583\",\"imageData\":
\"iVBORw0KGgoAAAANSUhEUgAAABkAAAAZCAYAAADE6YVjAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADs
QBlSsOGwAAAxVJREFUSInt1V1MU2cYwPF/YhRaCREiH5HVr3WgQUKyGDUab0S3i4Vkxl0Yw41ZRm
K2+HG9hMwtMdkFRi8gJkuWLCxzi1kmk4ABK4qiVJqWwaEIr9oK9NDT010QiaW2OC8WE32uznne9zy/5zk5H2
b+hzC/U4ilqVT+oswp7cgzZeVjwajGFiOjQTH+e0C9DVwGtDdGWjw1B9bYtDMNG4uqc7MtlqTF7XD8ebT3Yr
+/sfu
+uKCqau9/Rlqbyts2FFjaal1rc9Ptybdb61qbqj7c5AjmnPnVVwD88trIuf01B21Wvql1yVI6ILwQpW9smpF
/hCmscqi1sdTc/mfgEXB/VSQBhq817cIOlzMtAHCl38+lAQWArqP1FOVZ940oEeW3B
+L4qsj5I54vyyWKDBmA6GJsGQAotGVhgKwDm9dXDYWlWkVRhjIiT
+ZmP6t2rM80BHcnRPI1C4sU2q1UOmQPwrsXyIxIRpPLlGYMLa4TEBG
+6/Yl5aeezlNot2IykLOrXF7XOZzcRApiNpptL5/H4zpBEeHW
+AzdQ1P4hJqCz8diy8c2u2RPqbkyoaPFAWMCGJ0SdPYHueYTK7etaOylptSslBczBdF0ZvUExX1jIdp
+Hs5Y/EU48paa1xOgPJ8KrYqU5NuvTTyZ
+9T/OLn7UgnqqmSskkRH39KTJQFuh8SCppEAAjPPJm8K2xVIvqUpSFlBzqmeUXG4vqLENPtMZVGDbDPs2VKG
S87lYs8DAFo8Dpq3lTMYnObGw0l0HW4FZ7xCCO
+qkzR8e91/qcXT8XRe/fzEbnfS2o1HIdSoyvfNbqqdMgYD7KxwsrPCyWAgNP63L/DVynqvRACKO/qOhY
+4PxiZFNsqS+Tl/OZ1MvUb1qbsvzcpRP
+IctqrcOe1kY9BH27yfvTXH1sv94yGmt0u2Wi3WsiWkj/Ec9EYvf5pf2A6fLJzWPz0qlppEYCaY8Rh4JOuo1
vP3plQ2hO6qdZiRjZiNmo6kbmoFpyZf3z1h
+viFBBNVycj8iIO/zhwb2m4N4+35vf7HlmOfwEYGSjqIxnPkQAAAABJRU5ErkJggg==\",\"contentType
\":\"image/png\",\"color\" : null,\"width\":18.75,\"height\":18.75,\"angle\":0,
\"xoffset\":0,\"yoffset\":0}},{\"value\":\"4\",\"label\":\"Unknown or Other\",
\"description\":\"\",\"symbol\":{\"type\":\"esriPMS\",\"url\":\"D6B86A28\",
\"imageData\":
\"iVBORw0KGgoAAAANSUhEUgAAABsAAAAcCAYAAACQ0cTtAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADs
QBlSsOGwAAAwNJREFUSIntlktIVFEYx393Rj3OjGOOhVLmk8zQNmEa2CLKyt4PKnET05PRykIh0qxFYQ96SU
KWkJUtgooCKOMxlRb2oAiiIArptUhoMYpOo2fMua1mmqveGUuIFn2r7/7/3/l+9zucy7lh/MUI

```
107     String confirm = "not synced";
108     Integer[] arr;
109     SpatialReference webMercator = SpatialReference.create(102100);
110     QueryTask querytask = new QueryTask(
111             "http://gis.spatial.redlands.edu/ArcGIS/rest/services/melodi_king/events/
    FeatureServer/1");
112     com.esri.core.tasks.ags.query.Query query = new
    com.esri.core.tasks.ags.query.Query();
113     FeatureSet fset = new FeatureSet();
114     String filePath = "";
115     String syncResult;
116
117     @Override
118     protected void onCreate(Bundle savedInstanceState) {
119         // creates a new instance
120         super.onCreate(savedInstanceState);
121
122         // sets the content view to the "observation.xml" view in the layout
123         // folder
124         setContentView(R.layout.layout3);
125
126         // creates rating Bar object from view ID
127         ratingBar = (RatingBar) findViewById(R.id.ratingBar);
128
129         // rating bar listener listens for when the user changes the value
130         ratingBar.setOnRatingBarChangeListener(new OnRatingBarChangeListener() {
131             public void onRatingChanged(RatingBar ratingBar, float rating,
132                     boolean fromUser) {
133                 txtRating = String.valueOf(ratingBar.getRating());
134                 rateInt = Integer.valueOf((int) ratingBar.getRating());
135
136             }
137         });
138
139         // gets the x, y location from the intent data
140         locx = getIntent().getDoubleExtra("locx", 99999);
141         locy = getIntent().getDoubleExtra("locy", 99999);
142         userID = getIntent().getStringExtra("user");
143
144         mDbManager = new EventsDBManager(this);
145
146     }
147
148     public void onSaveClicked(View view) {
149         // save the event
150
151         if ((speciesPicked == false) || (countPicked == false)
152                 || calvesPicked == false) {
153             AlertDialog.Builder alert = new AlertDialog.Builder(this);
154
155             alert.setTitle("Fill required (bold) fields");
156
157             alert.setPositiveButton("Ok",
158                     new DialogInterface.OnClickListener() {
```

```
159                     public void onClick(DialogInterface dialog,
160                             int whichButton) {
161                         dialog.cancel();
162                     }
163                 });
164
165             alert.show();
166
167         } else {
168
169             saveEventEntry(syncResult);
170
171             // finish();
172
173         }
174
175     }
176
177     private void saveEventEntry(String syncResult) {
178         // TODO Auto-generated method stub
179
180         int eventType = 2;
181
182         // /
183         // /
184         // / shouldn't be hard coded!!
185
186         // Calendar c = Calendar.getInstance();
187         // SimpleDateFormat df = new SimpleDateFormat("MM/dd/YYYY HH:mm:ss");
188         String formattedDate = "9/29/2012";
189         Log.i("edits", "edits: " + catInt);
190         mDbManager.open();
191         // formattedDate have current date/time
192         mDbManager.createEventEntry(2, userID, locy, locx, catInt, speciesType,
193                 formattedDate, count, calves, behavior, notes, weatherString,
194                 seaString, rateInt, permissions, filePath);
195         mDbManager.close();
196         Log.i("edits", "done");
197         // Intent intent = getIntent();
198         // if (syncResult=="error"){
199         // setResult(RESULT_CANCELED);
200         // }
201         //
202         // if (syncResult=="pictureSuccess"){
203         // setResult(RESULT_FIRST_USER);
204         // }
205         // intent.putExtra("syncResult", syncResult);
206         finish();
207     }
208
209     public void onSyncClicked(View view) {
210
211         // creates a feature type object
212         FeatureType subType = new FeatureType();
```

```
213         subType = fLayer.getTypes()[catInt];
214
215         // calls the function that sends the new record to the server
216         applyEdits(GeometryEngine.project(locx, locy, webMercator), subType,
217                 fLayer, view);
218
219     }
220
221     public void applyEdits(Geometry geometry, FeatureType subType,
222             final ArcGISFeatureLayer featureLayer, View view) {
223
224         // create a graphic using the type
225         Calendar rightNow = Calendar.getInstance();
226
227         // creates graphic so that that the graphic can be sent to the server
228         Graphic graphic = featureLayer.createFeatureWithType(subType, geometry);
229
230         // adds attributes to the graphic
231         Map<String, Object> attr = graphic.getAttributes();
232         attr.put("Number", count);
233         attr.put("ObserverID", userID);
234         attr.put("Behavior", behavior);
235         attr.put("SpeciesType", speciesType);
236
237         int calfInt = 1;
238         if (calves == "No calves present") {
239             calfInt = 0;
240         }
241         attr.put("CalvesPresent", calfInt);
242         attr.put("Notes", notes);
243         attr.put("CloudCover", weatherString);
244         attr.put("Beafort", seaString);
245         attr.put("ConfidenceRating", rateInt);
246         attr.put("eventType", 1);
247         attr.put("Permissions", permissions);
248         attr.put("Date", rightNow);
249
250         // create a new graphic with the attributes. attributes are immutable
251         Graphic newGraphic = new Graphic(geometry, graphic.getSymbol(), attr,
252                 graphic.getInfoTemplate());
253
254         // applies edits
255         featureLayer.applyEdits(new Graphic[] { newGraphic }, null, null,
256                 new CallbackListener<FeatureEditResult[][]>() {
257
258                     public void onError(Throwable error) {
259
260                     }
261
262                     public void onCallback(FeatureEditResult[][] editResult) {
263                         Log.i("edits", "inside event callback");
264
265                         if (editResult[0] != null && editResult[0][0] != null
266                                 && editResult[0][0].isSuccess()) {
```

```
267                             syncResult = "success";
268                             Log.i("edits", syncResult);
269                             Intent intent = getIntent();
270
271                             Log.i("edits", syncResult);
272                             if (syncResult == "success") {
273
274                                 setResult(RESULT_OK);
275                             }
276
277                             if (filePath != "") {
278                                 int obj = editResult[0][0].getObjectId();
279
280                                 File fileP = new File(filePath);
281
282                                 featureLayer
283                                         .addAttachment(
284                                                 obj,
285                                                 fileP,
286                                                 new
    CallbackListener<FeatureEditResult>() {
287
288                                                     public void onCallback(
289                                                             FeatureEditResult objs)
    {
290                                                         // TODO Auto-generated
291                                                         // method stub
292                                                         if (objs.isSuccess() ==
    true) {
293                                                             syncResult =
    "pictureSuccess";
294                                                         }
295                                                         Log.i("edits",
296                                                                 "attachment
    callback");
297
298                                                     }
299
300                                                     public void onError(
301                                                             Throwable e) {
302                                                         // TODO Auto-generated
303                                                         // method stub
304
305                                                         Log.i("edits",
306                                                                 "attachment error");
307                                                         Log.i("edits",
308                                                                 e.getMessage());
309
310                                                     }
311
312                                                 });
313
314                             }
315                         }
```

```
316
317                     }
318
319                 });
320
321         saveEventEntry(syncResult);
322
323     }
324
325     // function that adds the attachment to the feature just sent to the server
326     public void addAttachment(int objectID) {
327         fLayer.addAttachment(objectID, new File(filePath),
328                 new CallbackListener<FeatureEditResult>() {
329
330                     public void onCallback(FeatureEditResult objs) {
331                         // TODO Auto-generated method stub
332                         Log.i("edits", "attachment callback");
333
334                     }
335
336                     public void onError(Throwable e) {
337                         // TODO Auto-generated method stub
338                         Log.i("edits", "attachment error");
339
340                     }
341
342                 });
343     }
344
345     public void results() {
346
347     }
348
349     public void behaviorSelected(View view) {
350         if (catPicked == false) {
351             Toast.makeText(getApplicationContext(), "please select a category",
352                     Toast.LENGTH_SHORT).show();
353         }
354         if (catPicked == true) {
355             AlertDialog.Builder btnCount = new AlertDialog.Builder(this);
356
357             final CharSequence[] countItems = { "Unknown", "Feeding",
358                     "Traveling", "Milling", "Logging", "Other" };
359
360             // modified from
361             //
   http://developer.android.com/guide/topics/ui/dialogs.html#CustomDialog
362             btnCount.setTitle("Describe behavior");
363
364             btnCount.setCancelable(true);
365
366             btnCount.setSingleChoiceItems(countItems, -1,
367                     new DialogInterface.OnClickListener() {
368                         public void onClick(DialogInterface dialog, int item) {
```

```
369                             // Toast.makeText(getApplicationContext(),
370                             // countItems[item], Toast.LENGTH_SHORT).show();
371                             behaviorSelected(countItems[item]);
372                             dialog.cancel();
373                         }
374
375                     });
376
377             AlertDialog alert = btnCount.create();
378             btnCount.show();
379
380         }
381
382     }
383
384     public void calvesSelected(View view) {
385         if (catPicked == false) {
386             Toast.makeText(getApplicationContext(), "please select a category",
387                     Toast.LENGTH_SHORT).show();
388         }
389         if (catPicked == true) {
390             AlertDialog.Builder btnCount = new AlertDialog.Builder(this);
391
392             final CharSequence[] countItems = { "No calves present",
393                     "One or more calves present" };
394
395             // modified from
396             //
    http://developer.android.com/guide/topics/ui/dialogs.html#CustomDialog
397             btnCount.setTitle("Are there calves present?");
398
399             btnCount.setCancelable(true);
400
401             btnCount.setSingleChoiceItems(countItems, -1,
402                     new DialogInterface.OnClickListener() {
403                         public void onClick(DialogInterface dialog, int item) {
404                             // Toast.makeText(getApplicationContext(),
405                             // countItems[item], Toast.LENGTH_SHORT).show();
406                             calvesSelected(countItems[item]);
407                             dialog.cancel();
408                         }
409
410                     });
411
412             AlertDialog alert = btnCount.create();
413             btnCount.show();
414
415         }
416     }
417
418     public void categorySelected(View view) {
419
420         // need to reset all variables here //
421         catPicked = true;
```

```
422         speciesPicked = false;
423         countPicked = false;
424         calvesPicked = false;
425         final TextView textViewToChange_count = (TextView)
    findViewById(R.id.btn_count);
426         textViewToChange_count.setText("Specify a quantity");
427
428         final TextView textViewToChange_calves = (TextView)
    findViewById(R.id.btn_calves);
429         textViewToChange_calves.setText("Presence of calves");
430
431         final TextView textViewToChange_behav = (TextView)
    findViewById(R.id.btn_behavior);
432         textViewToChange_behav.setText("Describe Behavior");
433
434         final TextView textViewToChange_photo = (TextView)
    findViewById(R.id.btn_photo);
435         textViewToChange_photo.setText("Add a photo");
436
437         final TextView textViewToChange_weath = (TextView)
    findViewById(R.id.btn_weather);
438         textViewToChange_weath.setText("Describe the weather");
439
440         final TextView textViewToChange_notes = (TextView)
    findViewById(R.id.btn_notes);
441         textViewToChange_notes.setText("Add a note");
442
443         switch (view.getId()) {
444         case R.id.btn_whale:
445
446             // from:
447             //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
448             final TextView textViewToChange_whale = (TextView)
    findViewById(R.id.btn_species);
449             textViewToChange_whale.setText("Select a whale species");
450             category = "Whale";
451             catInt = 1;
452
453             break;
454
455         case R.id.btn_dolphin:
456             // from:
457             //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
458             final TextView textViewToChange_dolph = (TextView)
    findViewById(R.id.btn_species);
459             textViewToChange_dolph
460                     .setText("Select a dolphin or porpoise species");
461             category = "Dolphin or Porpoise";
462             catInt = 2;
463             break;
```

```
464
465         case R.id.btn_seal:
466             // from:
467             //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
468             final TextView textViewToChange_seal = (TextView)
    findViewById(R.id.btn_species);
469             textViewToChange_seal.setText("Select a seal or sea lion species");
470             category = "Seal or Sea Lion";
471             catInt = 3;
472             break;
473
474         case R.id.btn_unknown:
475             // from:
476             //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
477             final TextView textViewToChange_unkn = (TextView)
    findViewById(R.id.btn_species);
478             textViewToChange_unkn.setText("Describe what you see");
479             category = "Unknown or Other";
480             catInt = 4;
481             break;
482         default:
483
484         }
485
486     }
487
488     public void selectCount(View view) {
489         if (catPicked == false) {
490             Toast.makeText(getApplicationContext(), "please select a category",
491                     Toast.LENGTH_SHORT).show();
492         }
493         if (catPicked == true) {
494             AlertDialog.Builder btnCount = new AlertDialog.Builder(this);
495
496             if (category != "Dolphin or Porpoise") {
497                 final CharSequence[] countItems = { "1", "2", "3 or more" };
498
499                 // modified from
500                 //
    http://developer.android.com/guide/topics/ui/dialogs.html#CustomDialog
501                 btnCount.setTitle("Specify a quantity");
502
503                 btnCount.setCancelable(true);
504
505                 btnCount.setSingleChoiceItems(countItems, -1,
506                         new DialogInterface.OnClickListener() {
507                             public void onClick(DialogInterface dialog, int item) {
508
509                                 countSelected(countItems[item]);
510                                 dialog.cancel();
```

```
511                             }
512
513                         });
514
515                 AlertDialog alert = btnCount.create();
516                 btnCount.show();
517
518             }
519
520             if (category == "Dolphin or Porpoise") {
521                 final CharSequence[] countItems = { "1-10", "10-50",
522                         "50 or more" };
523
524                 // modified from
525                 //
    http://developer.android.com/guide/topics/ui/dialogs.html#CustomDialog
526                 btnCount.setTitle("Specify a quantity");
527
528                 btnCount.setCancelable(true);
529
530                 btnCount.setSingleChoiceItems(countItems, -1,
531                         new DialogInterface.OnClickListener() {
532                             public void onClick(DialogInterface dialog, int item) {
533                                 // Toast.makeText(getApplicationContext(),
534                                 // countItems[item], Toast.LENGTH_SHORT).show();
535                                 countSelected(countItems[item]);
536                                 dialog.cancel();
537                             }
538
539                         });
540
541                 AlertDialog alert = btnCount.create();
542                 btnCount.show();
543
544             }
545
546         }
547     }
548
549     public void selectSpecies(View view) {
550         // from:
551         //
    http://stackoverflow.com/questions/5646418/how-to-go-about-multiple-buttons-and-oncl
    icklisteners
552         if (catPicked == false) {
553             Toast.makeText(getApplicationContext(), "please select a category",
554                     Toast.LENGTH_SHORT).show();
555         }
556         if (catPicked == true) {
557
558             // case R.id.btn_whale:
559             AlertDialog.Builder btnSpecies = new AlertDialog.Builder(this);
560
561             // if whale chosen
```

```
562             if (category == "Whale") {
563                 final CharSequence[] speciesItems = { "Unidentified",
564                         "Sperm Whale", "Gray Whale", "Fin Whale", "Sei Whale",
565                         "Blue Whale", "Humpback Whale", "Minke Whale" };
566
567                 // modified from
568                 //
    http://developer.android.com/guide/topics/ui/dialogs.html#CustomDialog
569                 btnSpecies.setTitle("Select a species");
570
571                 btnSpecies.setCancelable(true);
572
573                 btnSpecies.setSingleChoiceItems(speciesItems, -1,
574                         new DialogInterface.OnClickListener() {
575                             public void onClick(DialogInterface dialog, int item) {
576                                 // Toast.makeText(getApplicationContext(),
577                                 // speciesItems[item],
578                                 // Toast.LENGTH_SHORT).show();
579                                 speciesSelected(speciesItems[item]);
580                                 dialog.cancel();
581                             }
582
583                         });
584
585                 AlertDialog alert = btnSpecies.create();
586                 btnSpecies.show();
587
588             } // end whale category chosen
589
590             // if dolphin/porpoise is chosen
591             if (category == "Dolphin or Porpoise") {
592                 final CharSequence[] items = { "Unidentified", "Common",
593                         "Bottlenose", "Killer Whale", "Pacific White-Sided",
594                         "Nor Right Whale Dolphin", "Risso's" };
595
596                 // modified from
597                 //
    http://developer.android.com/guide/topics/ui/dialogs.html#CustomDialog
598                 btnSpecies.setTitle("Select a species");
599
600                 btnSpecies.setCancelable(true);
601
602                 btnSpecies.setSingleChoiceItems(items, -1,
603                         new DialogInterface.OnClickListener() {
604                             public void onClick(DialogInterface dialog, int item) {
605                                 // Toast.makeText(getApplicationContext(),
606                                 // items[item], Toast.LENGTH_SHORT).show();
607                                 speciesSelected(items[item]);
608                                 dialog.cancel();
609
610                             }
611                         });
612                 ;
613                 AlertDialog alert = btnSpecies.create();
```

```
614                 btnSpecies.show();
615
616             } // end dolphin/porpoise category chosen
617
618             // if seal/sea lion is chosen
619             if (category == "Seal or Sea Lion") {
620                 final CharSequence[] items = { "Unidentified", "CA Sea Lion",
621                         "Harbor Seal", "N. Fur Seal" };
622
623                 // modified from
624                 //
    http://developer.android.com/guide/topics/ui/dialogs.html#CustomDialog
625                 btnSpecies.setTitle("Select a species");
626
627                 btnSpecies.setCancelable(true);
628
629                 btnSpecies.setSingleChoiceItems(items, -1,
630                         new DialogInterface.OnClickListener() {
631                             public void onClick(DialogInterface dialog, int item) {
632                                 // Toast.makeText(getApplicationContext(),
633                                 // items[item], Toast.LENGTH_SHORT).show();
634                                 speciesSelected(items[item]);
635                                 dialog.cancel();
636
637                             }
638                         });
639                 ;
640                 AlertDialog alert = btnSpecies.create();
641                 btnSpecies.show();
642
643             } // end seal/sea lion category chosen
644
645             // if unknown is chosen
646             if (category == "Unknown or Other") {
647
648                 AlertDialog.Builder alert = new AlertDialog.Builder(this);
649
650                 alert.setTitle("What do you see?");
651
652                 // Set an EditText view to get user input
653                 final EditText input = new EditText(this);
654                 alert.setView(input);
655
656                 alert.setPositiveButton("Ok",
657                         new DialogInterface.OnClickListener() {
658                             public void onClick(DialogInterface dialog,
659                                     int whichButton) {
660
661                                 Editable value = input.getText();
662                                 speciesSelected(value);
663
664                             }
665                         });
666
```

```
667                 alert.setNegativeButton("Cancel",
668                         new DialogInterface.OnClickListener() {
669                             public void onClick(DialogInterface dialog,
670                                     int whichButton) {
671                                 // Canceled.
672                             }
673                         });
674
675                 alert.show();
676
677             } // end unknown lion category chosen
678
679         } // end "if catpicked = true"
680
681     }
682
683     public void speciesSelected(CharSequence items) {
684         // from:
685         //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
686         speciesPicked = true;
687         final TextView textViewToChange = (TextView) findViewById(R.id.btn_species);
688         textViewToChange.setText("Species selected: " + items);
689         speciesType = items.toString();
690     }
691
692     public void countSelected(CharSequence items) {
693         // from:
694         //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
695
696         countPicked = true;
697         count = items.toString();
698         final TextView textViewToChange = (TextView) findViewById(R.id.btn_count);
699         textViewToChange.setText("Quantity: " + items);
700     }
701
702     public void calvesSelected(CharSequence items) {
703
704         calvesPicked = true;
705         // from:
706         //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
707         final TextView textViewToChange = (TextView) findViewById(R.id.btn_calves);
708         textViewToChange.setText("" + items);
709         calves = items.toString();
710     }
711
712     public void behaviorSelected(CharSequence items) {
713         // from:
714         //
```

```
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
715         final TextView textViewToChange = (TextView)
    findViewById(R.id.btn_behavior);
716         textViewToChange.setText("Behavior: " + items);
717         behavior = items.toString();
718     }
719
720     public void photoSelected(CharSequence items) {
721         // from:
722         //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
723         final TextView textViewToChange = (TextView) findViewById(R.id.btn_photo);
724         textViewToChange.setText("Path: " + items);
725
726     }
727
728     public void seaSelected() {
729         // from:
730         //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
731         final TextView textViewToChange = (TextView) findViewById(R.id.btn_weather);
732         textViewToChange.setText("" + seaString + ", " + weatherString);
733
734     }
735
736     public void noteAdded(CharSequence items) {
737         // from:
738         //
    http://stackoverflow.com/questions/4768969/how-do-i-change-textview-value-inside-jav
    a-code
739         final TextView textViewToChange = (TextView) findViewById(R.id.btn_notes);
740         textViewToChange.setText("Note: " + items);
741         notes = items.toString();
742     }
743
744     public void addNote(View view) {
745         if (catPicked == false) {
746             Toast.makeText(getApplicationContext(), "please select a category",
747                     Toast.LENGTH_SHORT).show();
748         }
749         if (catPicked == true) {
750             AlertDialog.Builder alert = new AlertDialog.Builder(this);
751
752             alert.setTitle("Add a Note");
753
754             // Set an EditText view to get user input
755             final EditText input = new EditText(this);
756             alert.setView(input);
757
758             alert.setPositiveButton("Ok",
759                     new DialogInterface.OnClickListener() {
```

```
760                         public void onClick(DialogInterface dialog,
761                                 int whichButton) {
762                             Editable value = input.getText();
763                             noteAdded(value);
764                             // Toast toast =
765                             // Toast.makeText(getApplicationContext(), value,
766                             // 5000);
767                             // toast.show();
768                         }
769                     });
770
771             alert.setNegativeButton("Cancel",
772                     new DialogInterface.OnClickListener() {
773                         public void onClick(DialogInterface dialog,
774                                 int whichButton) {
775                             // Canceled.
776                         }
777                     });
778
779             alert.show();
780         }
781     }
782
783     public void selectSea(View view) {
784         if (catPicked == false) {
785             Toast.makeText(getApplicationContext(), "please select a category",
786                     Toast.LENGTH_SHORT).show();
787         }
788         if (catPicked == true) {
789             final CharSequence[] items = { "Sea is smooth- mirror like",
790                     "Light breeze", "Moderate Breeze", "Strong Breeze",
791                     "Unsure" };
792             AlertDialog.Builder builder = new AlertDialog.Builder(this);
793             builder.setTitle("What are the sea conditions like?");
794
795             builder.setCancelable(true);
796
797             builder.setSingleChoiceItems(items, -1,
798                     new DialogInterface.OnClickListener() {
799                         public void onClick(DialogInterface dialog, int item) {
800                             // Toast.makeText(getApplicationContext(),
801                             // items[item], Toast.LENGTH_SHORT).show();
802                             seaString = (String) items[item];
803                             dialog.cancel();
804                             selectWeather();
805                         }
806                     });
807             AlertDialog alert = builder.create();
808             builder.show();
809         }
810     }
811
812     // modified from
813     // http://developer.android.com/guide/topics/ui/dialogs.html#CustomDialog
```

```
814     public void selectWeather() {
815         final CharSequence[] items = { "Clear Skies", "Some clouds",
816                 "Half of the sky is covered in clouds", "Very cloudy", "Unsure" };
817         AlertDialog.Builder builder = new AlertDialog.Builder(this);
818         builder.setTitle("... and the cloud over?");
819 
820         builder.setCancelable(true);
821 
822         builder.setSingleChoiceItems(items, -1,
823                 new DialogInterface.OnClickListener() {
824                     public void onClick(DialogInterface dialog, int item) {
825                         // Toast.makeText(getApplicationContext(), items[item],
826                         // Toast.LENGTH_SHORT).show();
827                         weatherString = (String) items[item];
828                         seaSelected();
829                         dialog.cancel();
830                     }
831                 });
832         ;
833         AlertDialog alert = builder.create();
834         builder.show();
835     }
836 
837     public void selectPhoto(View view) {
838         // obtained from:
839         //
    http://stackoverflow.com/questions/2507898/how-to-pick-a-image-from-gallery-sd-card-f
    or-my-app-in-android
840 
841         final CharSequence[] items = { "Camera", "Gallery" };
842 
843         AlertDialog.Builder builder = new AlertDialog.Builder(this);
844         builder.setTitle("Attach from");
845         builder.setItems(items, new DialogInterface.OnClickListener() {
846             public void onClick(DialogInterface dialog, int item) {
847 
848                 if (items[item] == "Gallery") {
849                     Intent i = new Intent(
850                             Intent.ACTION_PICK,
851                             android.provider.MediaStore.Images.Media.EXTERNAL_CONTEN
    T_URI);
852                     startActivityForResult(i, SELECT_PICTURE);
853                 }
854                 //
    http://stackoverflow.com/questions/10165302/dialog-to-pick-image-from-gallery-or-fro
    m-camera
855                 if (items[item] == "Camera") {
856                     Intent takePicture = new Intent(
857                             MediaStore.ACTION_IMAGE_CAPTURE);
858                     startActivityForResult(takePicture, 0);// zero can be
859                                                            // replced with any
860                                                            // action code
861                 }
862 
```

```
863                 // Toast.makeText(getApplicationContext(), items[item],
864                 // Toast.LENGTH_SHORT).show();
865             }
866         });
867         AlertDialog alert = builder.create();
868         builder.show();
869
870     }
871
872     // obtained from:
873     //
    http://stackoverflow.com/questions/2507898/how-to-pick-a-image-from-gallery-sd-card-f
    or-my-app-in-android
874     protected void onActivityResult(int requestCode, int resultCode,
875             Intent imageReturnedIntent) {
876         super.onActivityResult(requestCode, resultCode, imageReturnedIntent);
877
878         if (resultCode == RESULT_OK) {
879             Uri selectedImage = imageReturnedIntent.getData();
880             String[] filePathColumn = { MediaStore.Images.Media.DATA };
881
882             Cursor cursor = getContentResolver().query(selectedImage,
883                     filePathColumn, null, null, null);
884             cursor.moveToFirst();
885
886             int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
887             filePath = cursor.getString(columnIndex);
888             // Toast.makeText(getApplicationContext(), filePath, 5000).show();
889             cursor.close();
890
891             Bitmap yourSelectedImage = BitmapFactory.decodeFile(filePath);
892
893             // Toast toast=Toast.makeText(this, filePath, 2000);
894             // toast.show();
895             photoSelected(filePath);
896         }
897     }
898
899     // handles actions to take when buttons are clicked
900     public void OnButtonClick(View view) {
901         // declares intent as an Intent
902         Intent intent;
903         // switches the view based on cases
904         switch (view.getId()) {
905
906         default:
907             // sets the intent to the "Main" Activity
908             intent = new Intent(this, Main.class);
909             intent.putExtra("newEvents", true);
910             intent.putExtra("catInt", catInt);
911             break;
912
913         }
914         // have to actually start the Activity, the intent object tells which
```

```
915         // Activity to start
916
917         startActivity(intent);
918         finish();
919
920     }
921
922 }
923
```

```
 1 package edu.gis.spatial.redlands.edu.Cohort21.melodi_king;
 2
 3 import java.sql.Date;
60
61 public class Main extends Activity {
62
63     // **Declare map
64     MapView map;
65
66     // Declare updateEvents toggle integer
67     int updateEvents = 0;
68
69     // creates a web mercator spatial reference object
70     SpatialReference webMercator = SpatialReference.create(102100);
71
72     // location variables
73     double locy;
74     double locx;
75
76     // creates Point object;
77     Point wgspoint;
78     final static double SEARCH_RADIUS = 2;
79
80     // variable for the username
81     String mEventUser;
82
83     // creates database object
84     private EventsDBManager mDbManager;
85
86     // variables that help handle when users rotate the device
87     // delete both
88     boolean saving;
89     boolean paused;
90
91     // creates editlayer object
92     // edit layer used to send new features to the server
93     ArcGISFeatureLayer editLayer = new ArcGISFeatureLayer(
94             "http://gis.spatial.redlands.edu/ArcGIS/rest/services/melodi_king/events/
   FeatureServer/1",
95             MODE.SNAPSHOT);
96
97     // variables for startactivity for result
98     // can delete
99     int syncSuccess = 0;
100    int requestCode;
101
102    // event layer definition, used to define subtypes for when events are
103    // rendered locally on the map
104    String layerDefinition = "{\"currentVersion\":10.02,\"id\":1,\"name\":\"events
   \",\"type\":\"Feature Layer\",\"displayField\":\"ObserverID\",\"description\":\"\",
   \"copyrightText\":\"\",\"relationships\":[],\"geometryType\":\"esriGeometryPoint\",
   \"minScale\":0,\"maxScale\":0,\"extent\":{\"xmin\":-14115658.1665,\"ymin
   \":3240128.2558,\"xmax\":-13061473.3222,\"ymax\":4321833.7614,\"spatialReference\":
   {\"wkid\":102113}},\"drawingInfo\":{\"renderer\":{\"type\":\"uniqueValue\",
```

\"field1\":\"SpeciesCategory\",\"field2\" : null,\"field3\" : null,\"fieldDelimiter
\":\", \",\"defaultSymbol\":{\"type\":\"esriSMS\",\"style\":\"esriSMSCircle\",
\"color\":[0,158,37,255],\"size\":4,\"angle\":0,\"xoffset\":0,\"yoffset\":0,
\"outline\":{\"color\":[0,0,0,255],\"width\":1}},\"defaultLabel\":\"\u003call other
values\u003e\",\"uniqueValueInfos\":[{\"value\":\"2\",\"label\":\"Dolphin or
Porpoise\",\"description\":\"\",\"symbol\":{\"type\":\"esriPMS\",\"url\":
\"B698AF73\",\"imageData\":\"iVBORw0KGgoAAAANSUhEUgAAABgAAAAZCAYAAAArK
+5dAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADsQBlSsOGwAAAwFJREFUSInllWtIk2EUx397t/mmstSZS
20i3iorDbbuYhQlkUFGYGFEd6w+FEVpRZFhSR
+EUAlBsQhaQh/EriSRYjUUwyS3LphamJdZ5BtUZi/u0pcarc1pQRB0Pj1wzvP/PeccznlU/GVT/RcAtZE1ux
NISQsiJFSFSpAZ/tSNtbeJmnbABNj/CJBN/towUVe4UF496zEN6rksJZQIt38bBc01lC5rFE0Vsiw3/xZgl3
j2mF6edSxZnq/po4ty9lHITQ9ACFMW7+D0giR5XtB59oYDtyYEyOPiOlGeXJDMfBHAzDUASjjIevbgYBSAqe
hJxKhMJyv7K1+o4lA38GIcgEsxQnmFkVUiwHsGqOUcABJdVHHYI1pLLPspI4OcrA6ahh5Ss9cvoFC8kR8kz9
E5sNNCHSaKfSXpNokeTpFFEXcCVrJ1ZldUU6rNZrOMCfggS1uSMHCPai5wxEtQSyKbOcx0jIShQ2aEFuqopp
g8KhZim5wJfgCBTIoRUFJLiZd4NvlkspMgND
+VQI2RFVRwgGc0BxpYob9Nx9glUqAMduJEwuYFSCDFQ/yHWTBzlCuEEEEwYcE/
+3w12fGeAWEG6Wwij1hmEkAgfbykHhORxBNFnDvaiZPpGJhKDACt3PMYOi+Ag9GhYT5GHqXK47WxJLOdMzh
+GVoBwS3uxMUQfYN
+ARFafYNLYpOvUihQoELtnfR366ez36Ktq0HyA4iWkk484s5GPUnKH6IALlzus51RnyArD9okSXriN4Mcpr0
u0ZovWSXzTjUiGrToSaSdRvTMYArRWHlIFPHoiKGeqxhYzjt637TTePxXPZ
+rYraUltvE7blveTVPSyQdtJFAKiN8wYqZSOKwY8eKmcWsoY36z328LHpCg3VCgAwUzkqeLVIimLp5uiGFJY
KGUDSEo/veUIBworFwv3OQziPXKav1pTXmNs1ltgPIqaTtdAetpeBMDSBQC4Lgwv5xmOEeSRy8e1k
+cRL4OpbOuB9OLobnQIZPpzze7X/ky/zPAd8AJlX1ZOTUiUwAAAAASUVORK5CYII=\",\"contentType\":
\"image/png\",\"color\" : null,\"width\":18,\"height\":18.75,\"angle\":0,\"xoffset
\":0,\"yoffset\":0}},{\"value\":\"0\",\"label\":\"None\",\"description\":\"\",
\"symbol\":{\"type\":\"esriPMS\",\"url\":\"C466E8E2\",\"imageData\":
\"iVBORw0KGgoAAAANSUhEUgAAAAwAAAAMCAYAAABWdVznAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADs
QBlSsOGwAAAJ1JREFUKJGt0bEKQVEYB/DfQDezDBZ5C6U8AQm7V7CIyWSUhzGZWL2EzAaZ9WWxUPderlLOeM
75nf7/75T8uEr/AHWs0Hue7zDH8ROo4oBmam
+ADlo4ZUCSJIuISF9OP7TGMAMiYvQlfheVfKT6F1BGLQ/OaBSAOy75DpuImBSALW75Dkv0ZacEV0x5H
+sVbdl/2GOG0yfw6jEuiOUB/Z0hYNqEvhsAAAAASUVORK5CYII=\",\"contentType\":\"image/png\",
\"color\" : null,\"width\":9,\"height\":9,\"angle\":0,\"xoffset\":0,\"yoffset\":0}},
{\"value\":\"3\",\"label\":\"Seal or Sea Lion\",\"description\":\"\",\"symbol\":
{\"type\":\"esriPMS\",\"url\":\"82D47583\",\"imageData\":
\"iVBORw0KGgoAAAANSUhEUgAAABkAAAAZCAYAAADE6YVjAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADs
QBlSsOGwAAAxVJREFUSInt1V1MU2cYwPF/cYwPF/vziFFkIOiJRaCREiH5HVr3WgQUKyGDUab0S3i4Vkxl0Yw41ZRm
K2+HG9hMwtMdkFRi8gJkuWLCxzi1kmk4ABK4qiVJqWwaEIr9oK9NDT010QiaW2OC8WE32uznne9zy/5zk5H2
b+hzC/U4ilqVT+oswp7cgzZeVjwajGFiOjQTH+e0C9DVwGtDdGWjw1B9bYtDMNG4uqc7MtlqTF7XD8ebT3Yr
+/sfu
+uKCqau9/Rlqbyts2FFjaal1rc9Ptybdb61qbqj7c5AjmnPnVVwD88trIuf01B21Wvql1yVI6ILwQpW9smpF
/hCmscqi1sdTc/mfgEXB/VSQBhq817cIOlzMtAHCl38+lAQWArqP1FOVZ940oEeW3B
+L4qsj5I54vyyWKDBmA6GJsGQAotGVhgKwDm9dXDYWlWkVRhjIiT
+ZmP6t2rM80BHcnRPI1C4sU2q1UOmQPwrsXyIxIRpPLlGYMLa4TEBG
+6/Yl5aeezlNot2IykLOrXF7XOZzcRApiNpptL5/H4zpBEeHW
+AzdQ1P4hJqCz8diy8c2u2RPqbkyoaPFAWMCGJ0SdPYHueYTK7etaOylptSslBczBdF0ZvUExX1jIdp
+Hs5Y/EU48paa1xOgPJ8KrYqU5NuvTTyZ
+9T/OLn7UgnqqmSskkRH39KTJQFuh8SCppEAAjPPJm8K2xVIvqUpSFlBzqmeUXG4vqLENPtMZVGDbDPs2VKG
S87lYs8DAFo8Dpq3lTMYnObGw0l0HW4FZ7xCCO
+qkzR8e91/qcXT8XRe/fzEbnfS2o1HIdSoyvfNbqqdMgYD7KxwsrPCyWAgNP63L/DVynqvRACKO/qOhY
+4PxiZFNsqS+Tl/OZ1MvUb1qbsvzcpRP
+IctqrcOe1kY9BH27yfvTXH1sv94yGmt0u2Wi3WsiWkj/Ec9EYvf5pf2A6fLJzWPz0qlppEYCaY8Rh4JOuo1
vP3plQ2hO6qdZiRjZiNmo6kbmoFpyZf3z1h
+viFBBNVycj8iIO/zhwb2m4N4+35vf7HlmOfwEYGSjqIxnPkQAAAABJRU5ErkJggg==\",\"contentType
\":\"image/png\",\"color\" : null,\"width\":18.75,\"height\":18.75,\"angle\":0,
\"xoffset\":0,\"yoffset\":0}},{\"value\":\"4\",\"label\":\"Unknown or Other\",
\"description\":\"\",\"symbol\":{\"type\":\"esriPMS\",\"url\":\"D6B86A28\",
\"imageData\":
\"iVBORw0KGgoAAAANSUhEUgAAABsAAAAcCAYAAACQ0cTtAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADs
OBlSsOGwAAAwNJREFUSIntlktIVFEYx393Rj3OiGOOhVLmk8zONmEa2CLKvt4PKnET05PRvkIh0qxFYO96Su

```
105
106     // creates featurelayer object
107     ArcGISFeatureLayer fLayer;
108
109     // creates cursor object
110     public Cursor mCursor;
111
112     // interval for handler
113     // private int m_interval = 300000; // 5 min by default, can be changed
114     // later
115     private int m_interval = 30000;
116
117     // creates handler object for repeating task
118     private Handler m_handler;
119
120     /** Called when the activity is first created. */
121     // @Override
122     public void onCreate(Bundle savedInstanceState) {
123         super.onCreate(savedInstanceState);
124         setContentView(R.layout.main);
125
126         // **Gets user ID from Home event
127         mEventUser = getIntent().getStringExtra("user");
128
129         // creates new map view from viewID
130         map = (MapView) findViewById(R.id.map);
131
132         // initiates database object
133         mDbManager = new EventsDBManager(this);
134
135         // adds the basemap
136     //  map.addLayer(new ArcGISDynamicMapServiceLayer(
137     //          "http://services.arcgisonline.com/ArcGIS/rest/services/World_Street_M
    ap/MapServer"));
138
139         map.addLayer(new ArcGISDynamicMapServiceLayer(
140                 "http://services.arcgisonline.com/ArcGIS/rest/services/Ocean_Basemap/
    MapServer"));
141
142         // editLayer isn't drawn, it is just used for sending position updates
143         // to the server
144         editLayer.setVisible(false);
145
146         map.addLayer(editLayer);
147
148         // calls the updatevents layer function
149         updateEventsLayers();
150
151         // ***Code snippet modified from Esri's Nearby Sample project
152         map.setOnStatusChangedListener(new OnStatusChangedListener() {
153
154             private static final long serialVersionUID = 1L;
155
156             // function that handles location changes, I got this function
```

```
157             // almost entirely from cope snippets from
158             // the ArcGIS Runtime SDK for Android "NearBy" Sample project
159             public void onStatusChanged(Object source, STATUS status) {
160                 // verifies that the map has been initialized
161                 if (source == map && status == STATUS.INITIALIZED) {
162
163                     LocationService ls = map.getLocationService();
164                     ls.setAutoPan(false);
165                     ls.setLocationListener(new LocationListener() {
166
167                         // boolean locationChanged = false;
168
169                         // Zooms to the current location when first GPS fix
170                         // arrives.
171
172                         public void onLocationChanged(Location loc) {
173                             locy = loc.getLatitude();
174                             locx = loc.getLongitude();
175                             wgspoint = new Point(locx, locy);
176
177                             Point mapPoint = (Point) GeometryEngine.project(
178                                     wgspoint, SpatialReference.create(4326),
179                                     map.getSpatialReference());
180
181                             Unit mapUnit = map.getSpatialReference().getUnit();
182                             double zoomWidth = Unit.convertUnits(SEARCH_RADIUS,
183                                     Unit.create(LinearUnit.Code.MILE_US),
184                                     mapUnit);
185
186                             // creates an envelope around users location and
187                             // zooms to the extent
188                             Envelope zoomExtent = new Envelope(mapPoint,
189                                     zoomWidth, zoomWidth);
190                             map.setExtent(zoomExtent);
191
192                         }
193
194                         public void onProviderDisabled(String arg0) {
195
196                         }
197
198                         public void onProviderEnabled(String arg0) {
199                         }
200
201                         public void onStatusChanged(String arg0, int arg1,
202                                 Bundle arg2) {
203
204                         }
205
206                     });
207
208                     // starts the location service listener
209                     ls.start();
210
```

```
211                 }
212
213             }
214         });
215
216         // ***End of code snippet modified from Esri's Nearby Sample project
217
218         // creates handler object
219         // can probably delete
220         m_handler = new Handler();
221
222         // starts the repeating task that updates the position every 5 minutes
223         startRepeatingTask();
224
225     }
226
227     // //////idea for handler/runnable from obtained from
228     //
    http://stackoverflow.com/questions/10207612/android-execute-code-in-regular-interval
    s
229     Runnable m_statusChecker = new Runnable() {
230         public void run() {
231
232             if (locx == 0.0) {
233                 Log.i("main", "locx is zero");
234
235             } else {
236                 // m_interval = 120000;
237                 onPositionUpdate(null); // this function can change value of
238                                         // m_interval.
239
240             }
241             // tells the task to repeat in m_interval minutes
242             m_handler.postDelayed(m_statusChecker, m_interval);
243         }
244     };
245
246     void startRepeatingTask() {
247         m_statusChecker.run();
248     }
249
250     void stopRepeatingTask() {
251         m_handler.removeCallbacks(m_statusChecker);
252     }
253
254     // handles rotation of phone.. so that data isn't reset/reloaded
255     // can delete because app was updated so that the orientation never changes
256     public void onConfigurationChanged(Configuration newConfig) {
257
258         super.onConfigurationChanged(newConfig);
259
260     }
261
262     // private class for handling each event record
```

```
263     // idea for this came from the "Feature Collection" example
264     static class Item {
265         private double latitude;
266         private double longitude;
267         private int category;
268         private int id;
269         private String type;
270         private int eventType;
271         private String user;
272         private String date;
273         private String count;
274         private String calves;
275         private String behavior;
276         private String notes;
277         private String cloud;
278         private String sea;
279         private int confidence;
280         private int permissions;
281 
282         public void setPermissions(int permissions) {
283             this.permissions = permissions;
284         }
285 
286         public void getPermissions(int permissions) {
287             this.permissions = permissions;
288         }
289 
290         public void setConfidence(int confidence) {
291             this.confidence = confidence;
292         }
293 
294         public void getConfidence(int confidence) {
295             this.confidence = confidence;
296         }
297 
298         public void setCloud(String cloud) {
299             this.cloud = cloud;
300         }
301 
302         public void getCloud(String cloud) {
303             this.cloud = cloud;
304         }
305 
306         public void setSea(String sea) {
307             this.sea = sea;
308         }
309 
310         public void getSea(String sea) {
311             this.sea = sea;
312         }
313 
314         public void setNotes(String notes) {
315             this.notes = notes;
316         }
```

```
317
318         public void getNotes(String notes) {
319             this.notes = notes;
320         }
321
322         public void setBehavior(String behavior) {
323             this.behavior = behavior;
324         }
325
326         public void getBehavior(String behavior) {
327             this.behavior = behavior;
328         }
329
330         public void setCalves(String calves) {
331             this.count = calves;
332         }
333
334         public void getCalves(String calves) {
335             this.count = calves;
336         }
337
338         public void setCount(String count) {
339             this.count = count;
340         }
341
342         public void getCount(String count) {
343             this.count = count;
344         }
345
346         public void setDate(String date) {
347             this.date = date;
348         }
349
350         public void getDate(String date) {
351             this.date = date;
352         }
353
354         public void setID(int id) {
355             this.id = id;
356         }
357
358         public int getID() {
359             return id;
360         }
361
362         public void setCatgeory(int category) {
363             this.category = category;
364         }
365
366         public int getCatgeory() {
367             return category;
368         }
369
370         public void setUser(String user) {
```

```
371             this.user = user;
372         }
373
374         public void setEventType(int eventType) {
375             this.eventType = eventType;
376         }
377
378         public double getEventType() {
379             return eventType;
380         }
381
382         public void setSpecies(String type) {
383             this.type = type;
384         }
385
386         public void getSpecies(String type) {
387             this.type = type;
388         }
389
390         public void getEventType(int eventType) {
391             this.eventType = eventType;
392         }
393
394         public void getUser(String user) {
395             this.user = user;
396         }
397
398         public double getLatitude() {
399             return latitude;
400         }
401
402         public void setLatitude(double latitude) {
403             this.latitude = latitude;
404         }
405
406         public double getLongitude() {
407             return longitude;
408         }
409
410         public void setLongitude(double longitude) {
411             this.longitude = longitude;
412         }
413
414     }
415
416     // called onCreate, grabs all of the observations in the database and
417     // renders them on the map
418     public void updateEventsLayers() {
419
420         // opens the database
421         mDbManager.open();
422
423         ArrayList<Item> items = new ArrayList<Item>();
424         ArrayList<Graphic> grList = new ArrayList<Graphic>();
```

```
425
426         mCursor = mDbManager.fetchAll();
427
428         // can probably delete, not being used
429         int i = 0;
430
431         mCursor.moveToFirst();
432
433         while (mCursor.isAfterLast() == false) {
434             Item item = new Item();
435
436             item.setID(mCursor.getInt(0));
437             item.setUser(mCursor.getString(1));
438             item.setEventType(mCursor.getInt(2));
439             item.setLatitude(mCursor.getDouble(3));
440             item.setLongitude(mCursor.getDouble(4));
441             item.setCatgeory((mCursor.getInt(5)));
442             item.setSpecies(mCursor.getString(6));
443             item.setDate(mCursor.getString(7));
444             item.setCount(mCursor.getString(8));
445             item.setCalves(mCursor.getString(9));
446
447             item.setBehavior(mCursor.getString(10));
448
449             item.setNotes(mCursor.getString(11));
450
451             item.setCloud(mCursor.getString(12));
452             item.setSea(mCursor.getString(13));
453
454             item.setConfidence(mCursor.getInt(14));
455             item.setPermissions(mCursor.getInt(15));
456
457             items.add(item);
458
459             // can probably delete, not being used
460             i = i + 1;
461             mCursor.moveToNext();
462         }
463
464         for (Item item : items) {
465
466             // creates a point for each item in the item array
467             Point pointwm = GeometryEngine.project(item.getLongitude(),
468                     item.getLatitude(), webMercator);
469             HashMap<String, Object> attrs = new HashMap<String, Object>();
470
471             // assigns the species category to the item
472             attrs.put("SpeciesCategory", item.getCatgeory());
473
474             // creates a new graphic with the x, y and attributes
475             // adds to the list of graphics that will be added to the map
476             Graphic gr = new Graphic(pointwm, null, attrs, null);
477
478             grList.add(gr);
```

```
479
480         }
481
482         Graphic[] grs = new Graphic[grList.size()];
483
484         grs = grList.toArray(grs);
485
486         FeatureSet fs = new FeatureSet();
487
488         // creates a feature set from the graphics
489         fs.setGraphics(grs);
490
491         Options options = new Options();
492         options.mode = MODE.SNAPSHOT;
493
494         // adds the feature layer to the map
495         try {
496
497             fLayer = new ArcGISFeatureLayer(layerDefinition, fs, options);
498
499             map.addLayer(fLayer);
500
501         } catch (Exception e) {
502             // TODO Auto-generated catch block
503             e.printStackTrace();
504         }
505
506         // closes the database
507         mDbManager.close();
508
509     }
510
511     // function that updates the position
512     public void onPositionUpdate(View view) {
513
514         // don't think i need to initiate a new object
515         // can probably delete
516         mDbManager = new EventsDBManager(this);
517
518         mDbManager.open();
519
520         // calls the mDbManager create event entry function and feeds in the
521         // required variables to create a new record in the database
522         mDbManager.createEventEntry(1, mEventUser, locy, locx, 0, "",
523                 "dateString", "", "", "", "", "", "", 0, 2, "");
524         mDbManager.close();
525
526         // lets the uesr know that the position has been updated
527         Toast toast = Toast.makeText(this, "position updated", 5000);
528         toast.show();
529
530         // calls the newEvents function so that new position update can be drawn
531         // on the map
532         newEvents();
```

```
533
534         // creates a feature type so that the new position update can be sent to
535         // the server
536         FeatureType subType = new FeatureType();
537
538         // gets the subtype of the feature being sent
539         subType = editLayer.getTypes()[0];
540
541         // calls the appleEdits function to send the feature to the server
542         applyEdits(GeometryEngine.project(locx, locy, webMercator), subType,
543                 editLayer);
544
545     }
546
547     public void applyEdits(Geometry geometry, FeatureType subType,
548             ArcGISFeatureLayer featureLayer) {
549
550         // creates a calendar object with current time
551         Calendar rightNow = Calendar.getInstance();
552
553         // creates string of the calendar object
554         String dateString = DateFormat.getDateTimeInstance().format(
555                 rightNow.getTime());
556
557         // creates a graphic to send to the server
558         Graphic graphic = featureLayer.createFeatureWithType(subType, geometry);
559
560         // gets attributes of the feature
561         Map<String, Object> attr = graphic.getAttributes();
562
563         // assigns the attributes to the feature
564         attr.put("ObserverID", mEventUser);
565
566         attr.put("ConfidenceRating", 0);
567         attr.put("eventType", 0);
568         attr.put("Permissions", 2);
569         attr.put("Date", rightNow);
570
571         // create a new graphic with the attributes. attributes are immutable
572         Graphic newGraphic = new Graphic(geometry, graphic.getSymbol(), attr,
573                 graphic.getInfoTemplate());
574
575         // applies edits
576         featureLayer.applyEdits(new Graphic[] { newGraphic }, null, null,
577                 new CallbackListener<FeatureEditResult[][]>() {
578
579                     public void onError(Throwable error) {
580                         Toast.makeText(getApplicationContext(), "error", 5000)
581                                 .show();
582                     }
583
584                     public void onCallback(FeatureEditResult[][] editResult) {
585
586                         if (editResult[0] != null && editResult[0][0] != null
```

```
587                             && editResult[0][0].isSuccess()) {
588                         Log.i("main", "inside call back");
589
590                     }
591
592                 }
593
594             });
595
596     }
597
598     // function that is respoinsible for adding new feature to the map
599     public void newEvents() {
600         mDbManager.open();
601
602         // populates the cursor with all of the records that havent been added
603         // to the map
604         mCursor = mDbManager.fetchUnapplied();
605
606         // don't think I need this variable anymore
607         // not being used
608         int i = 0;
609
610         // repeats for every item in the cursor
611         mCursor.moveToFirst();
612
613         if (mCursor.getCount() > 0) {
614
615             ArrayList<Item> items = new ArrayList<Item>();
616             ArrayList<Graphic> grList = new ArrayList<Graphic>();
617
618             while (mCursor.isAfterLast() == false) {
619                 Item item = new Item();
620
621                 // assigns
622                 item.setID(mCursor.getInt(0));
623                 item.setUser(mCursor.getString(1));
624                 item.setEventType(mCursor.getInt(2));
625                 item.setLatitude(mCursor.getDouble(3));
626                 item.setLongitude(mCursor.getDouble(4));
627                 item.setCatgeory((mCursor.getInt(5)));
628                 item.setSpecies(mCursor.getString(6));
629                 item.setDate(mCursor.getString(7));
630                 item.setCount(mCursor.getString(8));
631                 item.setCalves(mCursor.getString(9));
632
633                 item.setBehavior(mCursor.getString(10));
634
635                 item.setNotes(mCursor.getString(11));
636
637                 item.setCloud(mCursor.getString(12));
638                 item.setSea(mCursor.getString(13));
639
640                 item.setConfidence(mCursor.getInt(14));
```

```
641                 item.setPermissions(mCursor.getInt(15));
642
643                 mDbManager.UpdateApplied(item.getID());
644                 items.add(item);
645                 i = i + 1;
646                 mCursor.moveToNext();
647
648             }
649
650             for (Item item : items) {
651
652                 // creates a point for each item in the item array
653                 Point pointwm = GeometryEngine.project(item.getLongitude(),
654                         item.getLatitude(), webMercator);
655                 HashMap<String, Object> attrs = new HashMap<String, Object>();
656                 attrs.put("SpeciesCategory", item.getCatgeory());
657
658                 // creates a graphic that can be rendered
659                 Graphic gr = new Graphic(pointwm, null, attrs, null);
660                 Toast.makeText(getApplicationContext(),
661                         "" + item.getCatgeory(), Toast.LENGTH_SHORT).show();
662
663                 // adds graphic to the graphic list
664                 grList.add(gr);
665
666             }
667
668             Graphic[] grs = new Graphic[grList.size()];
669
670             grs = grList.toArray(grs);
671
672             FeatureSet fs = new FeatureSet();
673
674             // adds graphics to the feature set
675             fs.setGraphics(grs);
676
677             Options options = new Options();
678             options.mode = MODE.SNAPSHOT;
679             mDbManager.close();
680
681             try {
682                 // adds the feature set to the map
683                 fLayer = new ArcGISFeatureLayer(layerDefinition, fs, options);
684                 map.addLayer(fLayer);
685
686             } catch (Exception e) {
687                 // TODO Auto-generated catch block
688                 e.printStackTrace();
689             }
690
691         }
692
693         else {
694             Toast.makeText(getApplicationContext(), "no new events",
```

```
695                     Toast.LENGTH_SHORT).show();
696         }
697
698     }
699
700     // /no longer using onActivity for Result, can delete
701     @Override
702     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
703         // TODO Auto-generated method stub
704         Log.i("main", "inActivityResult");
705         super.onActivityResult(requestCode, resultCode, data);
706
707         if (data.getExtras().containsKey("syncResult")) {
708             Log.i("main", "synced");
709             Toast toast = Toast.makeText(this, "observation saved", 5000);
710             toast.show();
711         }
712         // if (resultCode == RESULT_CANCELED){
713         // Log.i("main", "synced");
714         // Toast toast = Toast.makeText(this, "not synced", 5000);
715         // toast.show();
716         // }
717
718         if (resultCode == RESULT_FIRST_USER) {
719             Log.i("main", "synced");
720             Toast toast = Toast.makeText(this, "photo saved", 5000);
721             toast.show();
722         }
723     }
724
725     public void OnButtonClick(View view) {
726
727         // creates intent
728         Intent intent;
729         switch (view.getId()) {
730         default:
731             intent = new Intent(this, Observation.class);
732             intent.putExtra("locx", wgspoint.getX());
733             intent.putExtra("locy", wgspoint.getY());
734             intent.putExtra("user", mEventUser);
735             updateEvents = 0;
736             break;
737
738         }
739
740         // starts new activity
741         startActivity(intent);
742
743         // pauses the Main activity
744         onPause();
745     }
746
747     // @Override
748     protected void onDestroy() {
```

```
749         super.onDestroy();
750
751         // stops recording new position updates
752         stopRepeatingTask();
753
754     }
755
756     // @Override
757     protected void onPause() {
758         // from Esri samples
759         super.onPause();
760         map.pause();
761
762     }
763
764     @Override
765     protected void onResume() {
766         super.onResume();
767
768         map.unpause();
769
770         // no longer being used, can delete
771         if (syncSuccess == 1) {
772             Toast toast2 = Toast.makeText(getApplicationContext(),
773                     "sync success", 5000);
774             toast2.show();
775         }
776         ;
777
778         // calls function that adds new events to the map
779         newEvents();
780
781     }
782
783     @Override
784     protected void onStop() {
785         // TODO Auto-generated method stub
786         super.onStop();
787     }
788
789 }
790
```

```java
package edu.gis.spatial.redlands.edu.Cohort21.melodi_king;

import java.util.ArrayList;

public class Home extends Activity {

    private String possibleEmail;
    public String mEventUser = "anonymous";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.home);

    }

    // Uses a pattern (emailPattern) to search through accounts that the user
    // has created
    public void setSettings(View view) {

        // find user's email address
        // Pattern emailPattern is a public class that you can use to verify
        // that a string is actually an email address

        Pattern emailPattern = Patterns.EMAIL_ADDRESS; // API level 8+

        // Account is a value type that represents an account in the
        // AccountManager
        Account[] accounts = AccountManager.get(this).getAccounts();
        List<String> listItems = new ArrayList<String>();
        listItems.add("anonymous");

        // Searches through all of the accounts that the user has created
        for (Account account : accounts) {
            if (emailPattern.matcher(account.name).matches()) {
                possibleEmail = account.name;
                listItems.add(possibleEmail);

            }

        }

        // let user choose which one
        // from:
        // http://stackoverflow.com/questions/7063831/android-how-to-populate-a-charsequence-array-dynamically-not-initializing
        // presents the user with a dialog box to choose which account they want
        // to associate with their survey
        final CharSequence[] items = listItems
                .toArray(new CharSequence[listItems.size()]);
        // end from

```

```
69          // final CharSequence[] items = {"Camera", "Gallery"};
70          AlertDialog.Builder builder = new AlertDialog.Builder(this);
71          builder.setTitle("Select a username");
72          builder.setItems(items, new DialogInterface.OnClickListener() {
73              public void onClick(DialogInterface dialog, int item) {
74
75                  mEventUser = items[item].toString();
76
77                  dialog.cancel();
78              }
79          });
80
81          // displays an alert box with all of the email accounts for the user to
82          // choose from
83          AlertDialog alert = builder.create();
84          builder.show();
85
86      }
87
88      public void OnButtonClick(View view) {
89
90          // creates intent for the main activity
91          Intent intent;
92          switch (view.getId()) {
93          default:
94
95              intent = new Intent(this, Main.class);
96              intent.putExtra("user", mEventUser);
97
98              break;
99
100         }
101         // starts Main activity
102         startActivity(intent);
103
104         // destroys the home activity
105         onDestroy();
106     }
107
108     @Override
109     protected void onDestroy() {
110         // TODO Auto-generated method stub
111         super.onDestroy();
112     }
113
114     @Override
115     protected void onPause() {
116         // TODO Auto-generated method stub
117         super.onPause();
118     }
119
120     @Override
121     protected void onRestart() {
122         // TODO Auto-generated method stub
```

```
123         super.onRestart();
124     }
125
126     @Override
127     protected void onResume() {
128         // TODO Auto-generated method stub
129         super.onResume();
130     }
131
132     @Override
133     protected void onStop() {
134         // TODO Auto-generated method stub
135         super.onStop();
136     }
137
138 }
139
```

```java
package edu.gis.spatial.redlands.edu.Cohort21.melodi_king.db;

import java.sql.Date;

public class EventsDBManager {
    public static final String DB_NAME = "eventsDB.db";
    public static final int SCHEMA_VERSION = 1;
    private Context mContext;
    private DBHelper mDbHelper;
    private SQLiteDatabase mDb;

    // Table fields
    public static final String SQL_CREATE_TABLE = "CREATE TABLE events("
            + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "OBSERVERID TEXT NOT NULL, " + "PHOTOPATH TEXT NOT NULL, "
            + "EVENTTYPE INTEGER NOT NULL, " + "latitude REAL NOT NULL, "
            + "longitude REAL NOT NULL, "
            + "SPECIESCATEGORY INTEGER NOT NULL, "
            + "SPECIESTYPE TEXT NOT NULL, " + "DATE TEXT NOT NULL, "
            + "COUNT TEXT NOT NULL, " + "CALVESPRESENT TEXT NOT NULL, "
            + "BEHAVIOR TEXT NOT NULL, " + "NOTES TEXT NOT NULL, "
            + "CLOUDCOVER TEXT NOT NULL, " + "BEAFORT TEXT NOT NULL, "
            + "CONFIDENCERATING INTEGER NOT NULL, "
            + "PERMISSIONS INTEGER NOT NULL," + "APPLYEDITS INTERGER NOT NULL"
            + ")";

    // old version of database schema
    // can delete
    // public static final String SQL_CREATE_TABLE = "CREATE TABLE events(" +
    // "_id INTEGER PRIMARY KEY AUTOINCREMENT, " + "user TEXT NOT NULL, " +
    // "latitude REAL NOT NULL, " + "longitude REAL NOT NULL" + ")";
    public static final String USER = "OBSERVERID";
    public static final String ID = "_id";
    public static final String LATITUDE = "latitude";
    public static final String LONGITUDE = "longitude";
    public static final String TABLE_NAME = "events";
    public static final String SPECIESCATEGORY = "SPECIESCATEGORY";
    public static final String SPECIESTYPE = "SPECIESTYPE";
    public static final String EVENTTYPE = "eventType";
    public static final String DATE = "DATE";
    public static final String COUNT = "COUNT";
    public static final String CALVESPRESENT = "CALVESPRESENT";
    public static final String BEHAVIOR = "BEHAVIOR";
    public static final String NOTES = "NOTES";
    public static final String CLOUDCOVER = "CLOUDCOVER";
    public static final String BEAFORT = "BEAFORT";
    public static final String CONFIDENCERATING = "CONFIDENCERATING";
    public static final String PERMISSIONS = "PERMISSIONS";
    public static final String APPLYEDITS = "APPLYEDITS";
    public static final String PHOTOPATH = "PHOTOPATH";

    private final class DBHelper extends SQLiteOpenHelper {

        public DBHelper(Context context) {
```

```
72              super(context, DB_NAME, null, SCHEMA_VERSION);
73              // TODO Auto-generated constructor stub
74          }
75
76          @Override
77          public void onCreate(SQLiteDatabase db) {
78
79              // creates new table, if needed
80              db.execSQL(SQL_CREATE_TABLE);
81
82          }
83
84          @Override
85          // required function
86          public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
87              // TODO Auto-generated method stub
88
89          }
90
91      }
92
93      public EventsDBManager(Context context) {
94          mContext = context;
95      }
96
97      public void open() {
98          mDbHelper = new DBHelper(mContext);
99          mDb = mDbHelper.getWritableDatabase();
100     }
101
102     public void close() {
103         mDbHelper.close();
104     }
105
106     // queries the specified table for all records, an returns the attributes
107     // specified for all attributes
108     // idea for this from Ref #1
109     public Cursor fetchAll() {
110         return mDb.query(TABLE_NAME, new String[] { ID, USER, EVENTTYPE,
111                 LATITUDE, LONGITUDE, SPECIESCATEGORY, SPECIESTYPE, DATE, COUNT,
112                 CALVESPRESENT, BEHAVIOR, NOTES, CLOUDCOVER, BEAFORT,
113                 CONFIDENCERATING, PERMISSIONS }, null, null, null, null, null);
114     }
115
116     // fetches all records in the database that haven't been rendered on the map
117     public Cursor fetchUnapplied() {
118         return mDb.query(TABLE_NAME, new String[] { ID, USER, EVENTTYPE,
119                 LATITUDE, LONGITUDE, SPECIESCATEGORY, SPECIESTYPE, DATE, COUNT,
120                 CALVESPRESENT, BEHAVIOR, NOTES, CLOUDCOVER, BEAFORT,
121                 CONFIDENCERATING, PERMISSIONS }, APPLYEDITS + " = 0", null,
122                 null, null, null, null);
123     }
124
125     // updates the applyedits field to 1. A value of 1 means the event has been
```

```
126     // rendered on the map.
127     public boolean UpdateApplied(int id) {
128         ContentValues values = new ContentValues();
129         values.put(ID, id);
130         values.put(APPLYEDITS, 1);
131 
132         return mDb.update(TABLE_NAME, values, ID + " = " + id, null) > 0;
133     }
134 
135     // create new event entry
136     public long createEventEntry(int eventType, String user, double latitude,
137             double longitude, int catInt, String Type, String date,
138             String count, String calves, String behavior, String notes,
139             String cloud, String sea, int confidence, int permissions,
140             String photopath) {
141 
142         // creates a ContentValue object and fills it with the attributes sent
143         // to it with the activity that called the function. Created a new
144         // record
145         // in the database and populates it with the incoming variable values
146         Log.i("database", "in create event");
147         ContentValues values = new ContentValues();
148         values.put(USER, user);
149         values.put(EVENTTYPE, eventType);
150         values.put(LATITUDE, latitude);
151         values.put(LONGITUDE, longitude);
152         values.put(SPECIESCATEGORY, catInt);
153         values.put(SPECIESTYPE, Type);
154         values.put(DATE, date);
155         values.put(COUNT, count);
156         values.put(CALVESPRESENT, calves);
157         values.put(BEHAVIOR, behavior);
158         values.put(NOTES, notes);
159         values.put(CLOUDCOVER, cloud);
160         values.put(BEAFORT, sea);
161         values.put(CONFIDENCERATING, confidence);
162         values.put(PERMISSIONS, permissions);
163         values.put(APPLYEDITS, 0);
164         values.put(PHOTOPATH, photopath);
165         return mDb.insert(TABLE_NAME, null, values);
166 
167     }
168 }
169 
170 // References:
171 // #1: The Android Videos that I bought (need to update)
```

Appendix C. Web Application HTML and Javascript Code

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
2 <html>
3
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
6     <title>whale mApp!
7     </title>
8
9 <script type="text/javascript">
10      var dojoConfig = {
11          parseOnLoad : true
12      };
13          </script>
14
15
16
17 <script type="text/javascript"
  src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.7"></script>
18  <link rel="stylesheet" type="text/css" href="layout1005.css">
19  <link rel="stylesheet" type="text/css"
  href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.7/js/dojo/dijit/themes/tundra/
  tundra.css"/>
20  <link rel="stylesheet" type="text/css"
  href="http://ajax.googleapis.com/ajax/libs/dojo/1.6/dojox/layout/resources/FloatingP
  ane.css"/>
21
22
23
24 <script src="myobs_scripts1005.js" type="text/javascript" ></script>
25
26   </head>
27
28 <body class="tundra">
29 <div id="header"><img id="loadingImg" src="header_400.png"/></div>
30 <br>
31 <br>
32
33 <div id="mainWindow" dojotype="dijit.layout.BorderContainer" design="headline"
  gutters="false" style="width:100%; height:100%;">
34
35
36 <!--TABS-->
37     <div dojotype="dijit.layout.TabContainer" id="tabcontain" region="center">
38
39 <!--start home tab-->
40 <div id="hometab" dojotype="dijit.layout.ContentPane" title = "Home"
  onShow="window.location.href='home.html';">
41 <!--end home tab-->
42 </div>
43
44
45 <!--start Map Tab-->
46 <div id="maptab" dojotype="dijit.layout.ContentPane"  title="Map"
```

```
  onShow="window.location.href='map.html';">
47 <!--End Map tab--!>
48 </div>
49
50 <!--START MyObs tab-->
51 <div id="myobs" dojotype="dijit.layout.ContentPane" title = "My Observations"
  selected="true">
52
53
54 <script type="dojo/method" data-dojo-event="onShow">
55 dijit.byId("forcelogin").show();
56 </script>
57
58
59
60 <!--Dialog box that tells user that they will be redirected-->
61 <div id="forcelogin" data-dojo-type="dijit.Dialog">
62 <table align="center">
63 <tr><td>
64 <label for="firstname">Enter your userID to get started: </label>
65
66 <input type="text" name="firstname" value="" dojoType="dijit.form.TextBox"
  id="userID"/>
67 </td></tr>
68
69 <!--When the myuser clicks the "take me there" button, eventually this will take
  users to Google to log in-->
70 <tr align="center"><td>
71 <button id="buttonEmail" type="button"></button>
72 </td></tr></table>
73 </div>
74
75
76
77
78
79 <!--START BORDER CONTAINER for mapAndNav -->
80 <div id="mapAndNav_border" dojotype="dijit.layout.BorderContainer" design="headline"
  gutters="false" style="width:100%; height:100%;">
81
82
83 <div id="mapAndNav_left" dojotype="dijit.layout.ContentPane" region="left">
84         <div dojotype="dijit.layout.AccordionContainer" >
85
86             <div id="down_accord" dojotype="dijit.layout.ContentPane"
  title="Download data" >
87
88                 <br>
89                 <font size="5"><i> Found what you're looking for?
90                     <br>
91                     <br>
92                     </i> Click "Extract Data" to get a <b>shapefile</b> of your
  selection
93                     <br>
```

```
                    <br>
                </font>
                <button dojoType="dijit.form.Button"  onClick="extractData();">
                    Extract Data
                </button>
                <br>
                <br>
                <img id="loadingImg" src="Load.gif" style="position:absolute;
left:40px;  z-index:100;display:none;" />

            </div>



<div id="submitNew" dojotype="dijit.layout.ContentPane" title="Submit Observations">


<div id="inputDiv">
<table cellpadding="5">
    <tr><td>
Date: <input type="text" name="date1" id="date1" value="2012-09-30"
    data-dojo-type="dijit.form.DateTextBox"
    required="true" />
</td></tr>
<tr>
    <td>
<label for="time1">
            Time:
        </label>
    <input type="text" name="time1" id="time1" value="T08:00:00"
dojoType="dijit.form.TimeTextBox"

onChange="dojo.byId('val').value = arguments[0].toString().replace(/.*1970\s(\S*).*/,'T
$1')"
        required="true" />

        <input readonly disabled id='val' value='T08:00:00'style="visibility:
hidden" />
       </td>
      </tr>
     </table>
</div>

<div id="inputDiv">
    <table cellpadding="5">
    <tr><td>
     <label for="lat">
            Latitude:
        </label>
        <input id="lat" type="text" dojoType="dijit.form.NumberTextBox"
name="latitude"
        value="" constraints="{min:-90,max:90,places:6}" required="true"
        invalidMessage="Invalid latitude">
```

```
142         </td>
143       </tr>
144
145       <tr>
146         <td>
147              <label for="long">
148             Longitude:
149         </label>
150         <input id="long" type="text" dojoType="dijit.form.NumberTextBox"
    name="longitude"
151         value="" constraints="{min:-180,max:180,places:6}"
152         invalidMessage="Invalid longitude">
153        </td>
154       </tr>
155
156 </table>
157 </div>
158
159 <div id="inputDiv">
160     <table cellpadding="5">
161 <tr>
162 <td>Species Category:
163     <select id="inputCat" onchange="CategoryChange()";>
164         <option value="Whale" selected='true'>
165             Whale
166             <option value="Dolphin or Porpoise">
167             Dolphin or Porpoise
168                 <option value="Seal or Sea Lion">
169             Seal or Sea Lion
170
171         </option>
172        </select>
173 </td>
174 </tr>
175
176 <tr>
177 <td>Species Type:
178
179     <select id="inputSpecies" >
180         <option value="Unidentified" selected='true'>
181             Unidentified
182         <option value="Blue">
183             Blue
184         <option value="Humpback">
185             Humpback
186         <option value="Gray">
187             Gray
188         <option value="Minke">
189             Minke
190         <option value="Sei">
191             Sei
192         <option value="Fin">
193             Fin
194         <option value="Sperm">
```

```
195             Sperm
196          </option>
197         </select>
198
199 </td>
200 </tr>
201
202 <tr>
203 <td id="inputcount">
204
205
206
207     Count: <select id="inputCount" > <option value="1" selected='true'>1<option
    value="2">2<option value="3 or more">3 or more</option> </select>
208
209
210
211 </td>
212 </tr>
213 <tr>
214     <td>
215 Presence of Calves:
216 <select id="inputCalves" >
217         <option value="No calves present" selected='true'>
218             No calves present
219             <option value="One or more calves present">
220             One or more calves present
221         </option>
222        </select>
223     </td>
224     <tr>
225     </table>
226
227 </div>
228
229
230 <div id="inputDiv">
231 <table cellpadding="5">
232     <tr>
233         <td>
234 Behavior:
235 <select id="inputBehavior" >
236         <option value="Unknown" selected='true'>
237             Unknown
238             <option value="Logging">
239             Logging
240             <option value="Milling">
241             Milling
242             <option value="Feeding">
243             Feeding
244             <option value="Traveling">
245             Traveling
246             <option value="Other">
247             Other
```

```
248         </option>
249        </select>
250        </td>
251       </tr>
252
253  <tr>
254  <td>
255 Seas:
256 <select id="inputSeas" >
257         <option value="Unsure" selected='true'>
258             Unsure
259             <option value="Sea is smooth- mirror like">
260             Sea is smooth- mirror like
261             <option value="Light breeze">
262             Light breeze
263             <option value="Moderate Breeze">
264             Moderate Breeze
265             <option value="Strong Breeze">
266             Strong Breeze
267         </option>
268        </select>
269        </td>
270       </tr>
271
272   <tr>
273  <td>
274 Weather:
275 <select id="inputWeather" >
276         <option value="Unsure" selected='true'>
277             Unsure
278             <option value="Clear Skies">
279             Clear Skies
280             <option value="Some clouds">
281             Some clouds
282             <option value="Half of the sky is covered in clouds">
283             Half of the sky is covered in clouds
284             <option value="Very cloudy">
285             Very cloudy
286
287         </option>
288        </select>
289        </td>
290       </tr>
291  <tr>
292  <td>
293 Confidence:
294 <select id="inputConfidence" >
295         <option value="1" selected='true'>
296             1
297             <option value="2">
298             2
299             <option value="3">
300             3
301             <option value="4">
```

```
302             4
303             <option value="5">
304             5
305         </option>
306        </select>
307        </td>
308       </tr>
309 </table>
310 </div>
311 <div id="inputDiv">
312 <table>
313     <tr>
314         <td>
315 <form id="attachment" enctype="multipart/form-data" >
316    <input name="uploadedfile" multiple="true" type="file"
    dojoType="dojox.form.Uploader" label="attachment" id="uploader"
    onChange="photoadded";/>
317
318 </form>
319 </td>
320
321 <td id="photoadded">
322     no photo added
323 </td>
324 </tr>
325 </table>
326 </div>
327  <button dojoType="dijit.form.Button"   onClick="submitObs();">
328                     Submit
329                 </button>
330
331
332     </div>
333
334
335 <div id="narrowSearch" dojotype="dijit.layout.ContentPane" title="Narrow your
    search" selected="true">
336     <br>
337 show results for:
338 <div id="showingFor"  >
339   <form name="selectData">
340         <table >
341         <tr>
342             <td><input type="radio" name="selectedItem" id="onlyMe" checked=false
    onchange="downloadQueryString()"/></td>
343             <td>Only me</td>
344
345             <td><input type="radio" name="selectedItem" id="allData" checked=true
    onchange="downloadQueryString()"/></td>
346             <td>Everyone</td>
347         </tr>
348
349               </table>
350
```

```
351
352
353     </form>
354
355
356 <!--end showingFor-->
357 </div>
358
359
360
361 date range:
362 <div id="accordianDate">
363
364 <table id="dateTable">
365
366 <tr>
367 <td>
368 from:
369 </td>
370 <td>
371 <!--list for users to specify a start month -->
372 <select id="monthBeg" onchange="dateChange()">
373             <option value="01" selected='true'>
374               Jan
375             </option>
376              ,
377             <option value="02">
378               Feb
379             </option>
380       <option value="03">
381               Mar
382             </option>
383 <option value="04">
384               Apr
385             </option>
386 <option value="05">
387              May
388             </option>
389 <option value="06">
390               Jun
391             </option>
392 <option value="07">
393               Jul
394             </option>
395 <option value="08">
396               Aug
397             </option>
398 <option value="09">
399               Sep
400             </option>
401 <option value="10">
402               Oct
403             </option>
404 <option value="11">
```

```
405              Nov
406            </option>
407 <option value="12">
408              Dec
409            </option>
410          </select>
411 </td>
412 <td>
413 <!--list for users to specify a start year -->
414 <select id="updateStart" onchange="dateChange()"; >
415        <option value="1999" selected='true'>
416              1999
417            </option>
418            <option value="2000">
419              2000
420            </option>
421      <option value="2001">
422              2001
423            </option>
424 <option value="2002">
425              2002
426            </option>
427 <option value="2003">
428              2003
429            </option>
430 <option value="2004">
431              2004
432            </option>
433 <option value="2005">
434              2005
435            </option>
436 <option value="2006">
437              2006
438            </option>
439 <option value="2007">
440              2007
441            </option>
442 <option value="2008">
443              2008
444            </option>
445 <option value="2009">
446              2009
447            </option>
448 <option value="2010">
449              2010
450            </option>
451  <option value="2011">
452              2011
453            </option>
454 <option value="2012">
455              2012
456            </option>
457          </select>
458
```

```
459 </td>
460 <tr>
461 <td>
462 to:
463 </td>
464 <td>
465 <!--list for users to specify an end month -->
466 <select id="monthEnd" onchange="dateChange()";>
467         <option value="01">
468               Jan
469             </option>
470
471             <option value="02">
472               Feb
473             </option>
474       <option value="03">
475               Mar
476             </option>
477 <option value="04">
478               Apr
479             </option>
480 <option value="05">
481              May
482             </option>
483 <option value="06">
484               Jun
485             </option>
486 <option value="07">
487               Jul
488             </option>
489 <option value="08">
490               Aug
491             </option>
492 <option value="09">
493               Sep
494             </option>
495 <option value="10">
496               Oct
497             </option>
498 <option value="11">
499               Nov
500             </option>
501 <option value="12" selected='true'>
502               Dec
503             </option>
504
505
506           </select>
507 </td>
508 <td>
509 <!--list for users to specify an end year -->
510 <select id="updateEnd" onchange="dateChange()"; >
511         <option value="1999" >
512               1999
```

```
513             </option>
514             <option value="2000">
515               2000
516             </option>
517       <option value="2001">
518               2001
519             </option>
520 <option value="2002">
521               2002
522             </option>
523 <option value="2003">
524              2003
525             </option>
526 <option value="2004">
527              2004
528             </option>
529 <option value="2005">
530               2005
531             </option>
532 <option value="2006">
533               2006
534             </option>
535 <option value="2007">
536               2007
537             </option>
538 <option value="2008">
539               2008
540             </option>
541 <option value="2009" >
542               2009
543             </option>
544             <option value="2010">
545               2010
546             </option>
547  <option value="2011">
548               2011
549             </option>
550 <option value="2012" selected='true'>
551               2012
552             </option>
553           </select
554 </td>
555 </tr>
556
557
558 </tr>
559 </table>
560 <font size="2"><i>
561 The months you choose will include observations for the entire month
562 <br>
563 </i></font>
564
565 <!--end date-->
566 </div>
```

```
567
568 event type:
569 <!--start event accordian container-->
570 <div id="accordianEvents">
571 <table>
572 <tr>
573 <td>
574 <input id="obsBox" ><label for="obsBox">observations</label>
575 </td>
576 </tr>
577 <tr>
578 <td>
579 <input id="trackBox" ><label for="trackBox">tracks</label>
580
581 </td>
582 </tr>
583 </table>
584
585
586 <!--end event accordian-->
587 </div>
588
589 species type:
590 <div id="accordianSpecies" >
591
592 <!--Table of species options-->
593 <table ><tr><td id = "species" >
594
595 <input id="whaleBox" ><label for="whaleBox">    whales   </label></td>
596 </tr>
597 <tr>
598 <td id = "species" >
599 <input id="dolphBox" ><label for="dolphBox">    dolphins and porpoises
    </label></td>
600
601 </tr>
602 <td id = "species" >
603 <input id="sealBox" ><label for="sealBox">  seals and sea lions  </label></td>
604
605 </tr>
606
607 </tr>
608 <td id = "species" >
609 <input id="otherBox" ><label for="otherBox">    other species  </label></td>
610
611 </tr>
612 </table>
613
614 <!--end species accordian-->
615 </div>
616
617
618
619
```

```
620 <!--end narrow-->
621 </div>
622
623
624
625
626 <!--start download accordian -->
627
628
629
630 <!--end of accordian container-->
631 </div>
632
633
634 <!--end of left border container for mapAndNav tab -->
635 </div>
636
637
638 <!--START AND END mapAndNav: MAP-- (CENTER)-->
639 <div id="map" dojotype="dijit.layout.ContentPane" region="center">
640
641 <!--
642 <div dojoType="dojox.layout.Dock" id="dock"></div>
643
644 <div dojoType="dojox.layout.FloatingPane" id="dFloatingPane" dockTo="dock"
    title="Identify"
645 resizable="true" closable="false" dockable="true" style="visibility:hidden;">
646
647   <table ><tr><td id="cellobsdetails_name" style="vertical-align:middle;
    text-align:left;" >
648   Click on an observation to learn more.
649   <br>
650   <br>
651   You can move this window around the screen and dock it for later by clicking on
    the arrow in the upper right hand corner.
652
653 </td></tr>
654 </td></tr></table>
655
656
657 </div>
658 -->
659
660
661
662 </div>
663
664
665
666
667
668
669 <!--END MyObs tab BORDER CONTAINER --!>
670 </div>
```

```
671 <!--END MyObs tab--!>
672 </div>
673
674
675
676
677 <!--start Learning Tab-->
678 <div id="learn" dojotype="dijit.layout.ContentPane"  title="Learn"
    onShow="window.location.href='learn.html';">
679 <!--END Learning Tab--!>
680 </div>
681
682
683
684 <!--END TAB CONTAINER --!>
685 </div>
686
687 <!--END BORDER CONTAINER--!>
688 </div>
689
690 </body>
691 </html>
```

```
dojo.require("dijit.dijit");
// optimize: load dijit layer
dojo.require("dijit.layout.BorderContainer");
dojo.require("dijit.layout.ContentPane");
dojo.require("esri.map");
dojo.require("dijit.layout.BorderContainer");
dojo.require("dijit.layout.ContentPane");
dojo.require("esri.graphic");
dojo.require("dijit.layout.TabContainer");
dojo.require("esri.tasks.query");
dojo.require("esri.dijit.TimeSlider");
dojo.require("dijit.Dialog");
dojo.require("dijit.form.Button");
dojo.require("dijit.form.CheckBox");
dojo.require("esri.layers.FeatureLayer");
dojo.require("dijit.layout.AccordionContainer");

dojo.require("dijit.form.TimeTextBox");
dojo.require("dijit.form.NumberTextBox");
dojo.require("dijit.form.TextBox");
dojo.require("dijit.form.DateTextBox");
dojo.require("dojox.layout.FloatingPane");
dojo.require("esri.dijit.InfoWindow");
dojo.require("dojox.form.Uploader");

//this global object contains all references needed across functions
//i dont understand how globals help, but it ensures that the map is filled to the full center extent

var query, queryTask;
var featureSet;
var map;
var attrib;
var eventlist;
var obsLayer;
var tracksLayer;
var whaleCheck = true;
var dolphinCheck = true;
var sealCheck = true;
var otherCheck = true;
var obsCheck = true;
var trackCheck = false;
var PinCheck = true;
var visibleObs = [];
var queryWhere = "";
var updateStart = "1999";
var updateEnd = "2012";
var updateStartDay;
var updateEndDay;
var monthBeg = 01;
var monthEnd = 12;
//var downloadDate = "\"DATE\" >= '" + updateStart +"-" + monthBeg + "-01' AND \"DATE\" <= '" + updateEnd + "-" + monthEnd + "-31'";
var downloadDate = "DATE>='" + updateStart + "-" + monthBeg + "-01' AND DATE<='" +
```

```
  updateEnd + "-" + monthEnd + "-31'";
53 var gp;
54 var loading;
55 var queryWhere;
56 var downloadQuery = "(SpeciesCategory = 1 OR SpeciesCategory = 2 OR SpeciesCategory
   = 3 OR SpeciesCategory = 4)";
57 var downloadSpecies = "(SpeciesCategory = 1 OR SpeciesCategory = 2 OR
   SpeciesCategory = 3 OR SpeciesCategory = 4)";
58 var downloadEvents = "eventType = 1";
59 var user;
60 var inputCat = "Whale";
61 var intCat = 1;
62
63 //var downLoadEvents = "eventType = 0 OR (";
64
65 //calls the init function @ the top of the doc on load
66 dojo.addOnLoad(init);
67
68 ////////////////////
69 //function is initialized on page's load
70 function init() {
71
72     //define a "loading" image for perceived progress while the data is being
   extracted
73     loading = dojo.byId("loadingImg");
74
75     //defines the initial extent of the map
76     //var initialExtent = new
   esri.geometry.Extent({"xmin":-14091002.812700,"ymin":3508363.61690,"xmax":-13252801.
   136300,"ymax":4388898.711300,"spatialReference":{"wkid":102100}});
77     var initialExtent = new esri.geometry.Extent({
78         "xmin" : -13826939.3720895,
79         "ymin" : 3543866.13823468,
80         "xmax" : -12631549.4813098,
81         "ymax" : 4216967.48443738,
82         "spatialReference" : {
83             "wkid" : 102100
84         }
85     });
86     map = new esri.Map("map", {
87         extent : initialExtent,
88
89     });
90
91     //expands the map so that it fills the full extent
92     dojo.connect(map, 'onLoad', function(map) {
93
94         dojo.connect(dijit.byId('map'), 'resize', resizeMap);
95     });
96
97     //this listener calls eventClicked
98     dojo.connect(map, 'onClick', eventClicked);
99
100    //declares the service location of the basemap
```

```
101     var basemap = new
    esri.layers.ArcGISTiledMapServiceLayer("http://server.arcgisonline.com/ArcGIS/rest/s
    ervices/Ocean_Basemap/MapServer");
102     map.addLayer(basemap);
103
104     //declares the map service location of the events layer
105     obsLayer = new
    esri.layers.FeatureLayer("http://gis.spatial.redlands.edu/ArcGIS/rest/services/melod
    i_king/events/FeatureServer/1", {
106         mode : esri.layers.FeatureLayer.MODE_SNAPSHOT
107     });
108
109     //only displays events of type 2 (these are observations)
110     obsLayer.setDefinitionExpression("eventType = 1");
111
112     //adds observations to the lap
113     map.addLayer(obsLayer);
114
115     //var selectionSymbol = new esri.symbol.SimpleFillSymbol().setColor(new
    dojo.Color([255,255,0,0.5]));
116     //obsLayer.setSelectionSymbol(selectionSymbol);
117     //declares the map service location of the events layer
118     //tracksLayer = new
    esri.layers.FeatureLayer("http://gis.spatial.redlands.edu/ArcGIS/rest/services/melod
    i_king/eventsFinal/MapServer//4", {
119     //          mode: esri.layers.FeatureLayer.MODE_SNAPSHOT});
120
121     //only displays events of type 2 (these are observations)
122     //tracksLayer.setDefinitionExpression("eventType = 2");
123
124     //adds observations to the lap
125     //map.addLayer(tracksLayer);
126
127     //declares new gp service
128     gp = new
    esri.tasks.Geoprocessor("http://gis.spatial.redlands.edu/ArcGIS/rest/services/melodi_
    king/events/GPServer/eventExtract");
129
130     //check boxes for types of marine mammals
131     //whale checkbox
132     var whaleBox = new dijit.form.CheckBox({
133         name : "whaleBox",
134         value : "agreed",
135         checked : true,
136         onChange : function(b) {
137             if (b == 1) {
138                 whaleCheck = true;
139
140                 //calls the updateVisibleSpecies function
141                 updateVisibleSpecies()
142
143             } else {
144                 whaleCheck = false;
145
```

```
146                 //calls the updateVisibleSpecies function
147                 updateVisibleSpecies()
148 
149             }
150         }
151     }, "whaleBox");
152 
153     //dolphin checkbox
154     var dolphBox = new dijit.form.CheckBox({
155         name : "dolphBox",
156         value : "agreed",
157         checked : true,
158         onChange : function(b) {
159             if (b == 1) {
160                 dolphinCheck = true;
161 
162                 updateVisibleSpecies()
163 
164             } else {
165                 dolphinCheck = false;
166 
167                 //calls the updateVisibleSpecies function
168                 updateVisibleSpecies()
169             }
170         }
171     }, "dolphBox");
172 
173     //seal and sea lion checkbox
174     var sealBox = new dijit.form.CheckBox({
175         name : "sealBox",
176         value : "agreed",
177         checked : true,
178         onChange : function(b) {
179             if (b == 1) {
180                 sealCheck = true;
181 
182                 updateVisibleSpecies()
183 
184             } else {
185                 sealCheck = false;
186 
187                 //calls the updateVisibleSpecies function
188                 updateVisibleSpecies()
189             }
190         }
191     }, "sealBox");
192 
193     //other checkbox
194     var otherBox = new dijit.form.CheckBox({
195         name : "otherBox",
196         value : "agreed",
197         checked : true,
198         onChange : function(b) {
199             if (b == 1) {
```