Reporting System for Road Kill Utilizing Mobile Devices

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

Of the Requirements for the Degree

Masters of Science in Computer Science

By

Guadalupe Ojeda

August 2012

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

[illegible] 8/20/12

Advisor: Dr. Andrzej Bieszczad Date

[illegible] 8/20/12

Dr. Sean Anderson Date

[illegible] 8/20/12

Dr. Peter Smith Date

APPROVED FOR THE UNIVERSITY

[illegible] 8-20-12

Dr. Gary A. Berg Date

Reporting System for Road Kill Utilizing Mobile Devices
Title of Item

Mobile Device, Road Kill, Wildlife-Vehicle Collision, Reporting System
3 to 5 keywords or phrases to describe the item

Guadalupe Ojeda
Author(s) Name (Print)

[illegible]
Author(s) Signature

08/22/2012
Date

Reporting System for Road Kill Utilizing Mobile Devices

By
Guadalupe Ojeda

Computer Science Program
California State University Channel Islands

Abstract

Living in an expanding industrialized society requires the creation and maintenance of functional roadways that meet the needs of a mobile population. Roads are essential for commuters to travel to their desired destinations, yet they may be constructed anywhere and everywhere without sufficient thought about possible consequences to the surrounding environment. As vehicles travel on current roadways, they may be killing surrounding wildlife on a daily basis. This destruction of wildlife, known as road kill, is an increasingly common problem that must be addressed. Urban dwellers need to know that the surrounding animal environment is an important consideration when roads are constructed or improved.

The overall goal of this project was to assist in creating a repository of data related to road kill. Researchers could gather information and formulate findings from the database through web-based data mining tools that are implemented now and in the future for the project. Development planners could then present their findings to minimize the negative impact of urban development upon wildlife and humans.

The implementation goal of this project was to design and develop a functional mobile application that could conduct transect observations using a Global Positioning System. Another aspect of this undertaking was to permit crowd-sourcing road kill information through an iPhone device that utilizes an Application Program Interface for reporting observations to a remote database.

Acknowledgements

I would like to thank Rebekah Ojeda and Daniel Ojeda for their support with my thesis project. I would like to thank my mother Marisela Chávez for her support during my thesis project. I would also like to thank Dr. Andrzej Bieszczad, Dr. Sean Anderson, and Dr. Peter Smith for their participation.

Table of Contents

Table of Figures

Chapter 1: Introduction

1.1 Introduction to the Problem and Solution for Road Kill

Commuters and city planners need to contemplate the impact of new and improved roadways upon the surrounding environment and wildlife. Road kill occurs when a vehicle kills an animal on a road that is utilized by the driving public. Many drivers are unaware of the impact of road kill upon the animal population. Data is not currently being collected because there is no central repository that would assemble such information. Without the data, city planners are not able to analyze the statistics in order to improve the roads for both animals and humans. Research is needed to quantify the severe roadway problems that arise with a growing industrialized population. There needs to be a way to track animal characteristics and location while traveling the various roads.

The Splatter Spotter application created for this study permits drivers to become more attentive to how a developing city disrupts various types of animal corridors. This application allows any user to gather and store road kill data to one central location. It also allows cities and other agencies to analyze the data collected by citizens in order to reduce the hazards of an animal presence upon the highways of a growing population.

The goal is to permit drivers to record their observations on various types of mobile devices. Commuters could gather useful data, including types of animals killed and the conditions of the surrounding environment, and enter it into their mobile phone application. The collected data could then be sent to a repository, where the information is stored and examined. Researchers and city planners could then evaluate the data and make adjustments to the roads to demonstrate consideration for wildlife in the surrounding areas and to decrease wildlife-vehicle collisions.

1.2 An Introduction to the Splatter Spotter Architecture

Splatter Spotter is an application that permits a commuter to do more than just ignore road kill. This application allows a user to record various observations regarding road kill. Road kill reports are collected, stored, and analyzed using various tools. Figure 1.1 describes the architecture of how various tasks are conducted for the Splatter Spotter application.

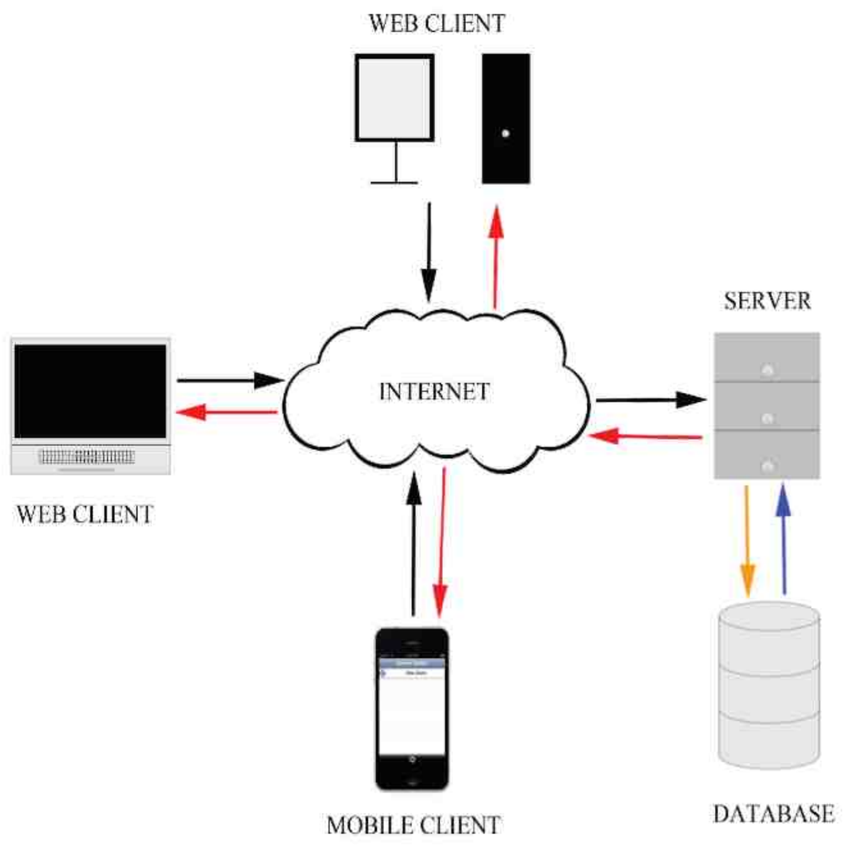


Figure 1.1 An Introduction to the Splatter Spotter Architecture

Figure 1.1 displays various elements that interact with each other to collect, store, and analyze reports. The elements shown in the image consist of server, database, mobile client, and web client.

A server is a machine in the cloud that accepts reports and delegates commands based on a set of rules provided by the application. The server allows a web client to access the web-based application in the cloud. Also, the app permits the clients to communicate with the database.

The database stores reports from various web and mobile clients. The database provides a repository for reporters to input information into one central location. Also, researchers are able to analyze the information stored in the database.

Mobile clients utilize mobile applications that interact with the server and database. Reporters upload their observations to the server situated in the cloud, represented as a black arrow in the Figure. Reports consist of data about the animal, location, optional image, and any additional information about the observation. After the server has validated the reports, then the server stores the reports into the database, represented as an orange arrow in the Figure. After the data has been stored into the database, a confirmation from the server is sent back to the reporter, represented as a red arrow. If the server has received an inaccurate report by the reporter, then an error message is sent back to the mobile client for correction, represented as a red arrow in the Figure.

Web clients run on laptop and desktop computers, which interact with the server and database through a web browser. Reporters upload their observations to the server situated in the cloud, represented as a black arrow in the Figure. After the server has validated the reports, then the server stores the reports into the database, represented as an orange arrow in the Figure. After the data has been stored into the database, a confirmation from the server is sent back to the reporter, represented as a red arrow. If the server has received an incorrect report by the reporter, then an error message is sent back to the web client for correction, represented as a red arrow in the Figure. Researchers use filters provided for the web client to access the data from the database. The black arrow is a query from the researcher to examine a specific area, animal, and so on. The orange arrow is the query for the database. The blue arrow is the retrieved data from the database based on the query. The red arrow represents sending the retrieved information to the researcher from the database.

1.3 Introducing the Mobile Client Application

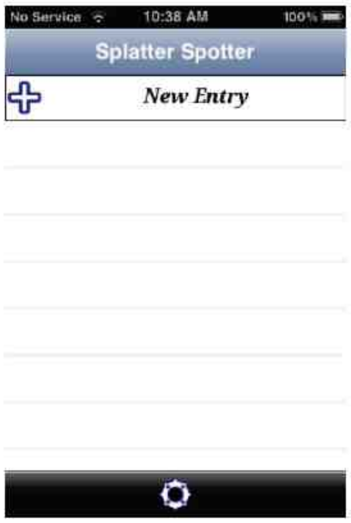


Figure 1.2 Introducing the Mobile Client Application

This thesis focuses on the mobile client-side of the project, which contains features that permit the commuters to report road kill, as shown in Figure 1.2. The tool permits the commuter to report a single or transect observation. A single report acquires a location, information about an animal, an image, and any additional information about the observation. A transect report allows for multiple single reports to be collected when a commuter drives for a lengthier segment of road. The mobile client application permits the commuter to manipulate any record locally by storing, deleting, viewing, and continuing any report. The server utilizes a custom Application Program Interface (API) to process commands received by the mobile client. Rules from the API allow the client-side to upload accurate information into the remote database.

1.4 Associating the Mobile Device with a Server

Accumulating data on a mobile client is convenient, but those reports are only available to a single person. A superior method is the ability to share every report stored on mobile client devices with others. A server is a perfect solution because all of the reports are situated in one central location and from there can be shared with others.

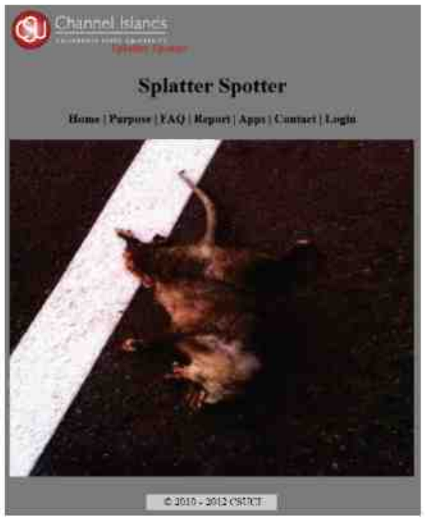


Figure 1.3 Associating the Mobile Device with a Server

The server executes additional functions that the app cannot perform because of the constraints of a mobile platform. The server-side contains an API, which allows mobile clients to communicate with the server. The app has a custom API that contains functions to gather photos taken from the mobile client and submit reports that were observed. User input is validated on the server-side, so that information sent from the mobile device is accepted. After the server has flagged the accepted information, the report is inserted into the remote database. The reports gathered in the database can be analyzed and visually overlaid onto a visual displaying tool, with the use of filters provided by the web-based application. The information is visually displayed as a single or transects reports that could be uploaded through the mobile or web clients. There are other features explained in greater detail in Ojeda's paper [26] concerning how the mobile client communicates to the server and how web clients analyze and store information from the database. Figure 1.3 displays the web-based application for the Splatter Spotter project.

1.5 A Summary of the Remaining Chapters

Chapter two is a field overview of the mobile device application.

Chapter three describes functions utilized for the mobile application.

Chapter four is an in-depth description of the design and functionalities of the application.

Chapter five states the conclusion of the thesis.

Chapter six provides suggestions for any future work that can be done or work that was not completed for this project.

1.6 Key Terms

API – Application Programming Interface

XML – Extensible Markup Language

SQLite – Structured Query Language Lite

iOS – Apple's mobile operating system

Wi-Fi – Wireless connectivity of electronic devices

3GS – Speedier 3rd generation mobile telecommunications

App – Application

Xcode – Apple's application development software

iPhone – Apple's Internet and multimedia-enabled smartphone

Transect – Recorded occurrences along a path

Waypoints – Collection of coordinates along a route defined by latitude and longitude

HTTP – Hypertext Transfer Protocol

Chapter 2: Field Overview

This chapter discusses the specific devices and the methodology needed to collect relevant road kill data. An explanation is provided about how information is being gathered and stored. The communication process is explained as to how the information is sent from the mobile client to server/database. Various tools are discussed in providing different ways of implementing mobile applications. Related apps are presented which address environmental issues.

2.1 Using Technology and Crowd Source to Address Environmental Challenges

Research indicates that motorists would voluntarily report road kill observations on their daily commute if given appropriate technology. Such technology could be delivered through web-based applications [20] and mobile devices with built in GPS software [27] to provide accuracy [19] for data collection. After supplying data to researchers, the results that come from studying the data would assist the government in constructing wildlife crossings [20] or mending existing roads. This project was implemented for the purpose of improving the environment by reporting road kill through a mobile device.

Crowdsourcing is the process of distributing an application to an unknown set of volunteers to solve a problem [40]. A crowdsourcing method permits numerous individuals to independently input road kill observations. This application provides individuals with a set of tools for the purpose of improving roads. Crowdsourcing also provides the developer with user feedback to improve the app for a better user experience.

2.2 The Role of Database

A database is a collection of digital data that is stored on the server [41] or directly to disk. A database is utilized on the mobile device to permit users to store past and present data. The database stores observation data, which are collected from the road kill. Collecting historical data is important for this project, as analyzers need to evaluate data from numerous reports. The important information that needs to be collected is broken down into the following categories: Time/Place, Road, Kill, and Additional. The different categories were selected because a data analyzer needs to know where and when the animal was killed, the composition of the road, what type of animal was killed, and any additional information that may help in analyzing the data. Using the above categories for this project, reporters can provide data analyzers with detailed reports that can be used to construct better roadways or modify existing ones.

2.3 The Communication Process between the Mobile Client and Server/Database

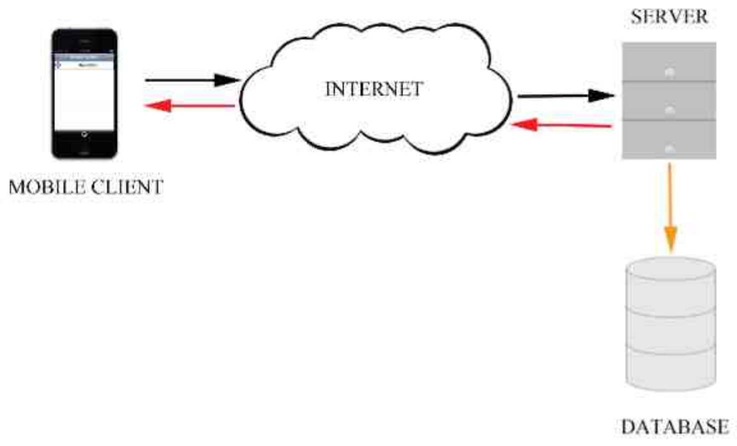


Figure 2.1 The Communication Process between Mobile Client and Server/Database

The communication protocol is a set of rules that allows a message to be communicated to the receiver [39]. This project contains a custom API that defines how the mobile client communicates with the server and database, as shown in Figure 2.1. An HTTP connection is established when a reporter uploads an observation to the server situated in the cloud, represented as a black arrow in the Figure. The parameters in the report are sent as a string in plain text to the server. After the server has validated the report, then the server stores the report into the remote database, represented as an orange arrow in the Figure. Once the data is stored into the database, a confirmation from the server is sent back to the reporter, represented as a red arrow. If the server has received an inaccurate report by the reporter, then an error message is sent back to the mobile client for correction, represented as a red arrow in the Figure. The confirmation and error messages communicate through the same HTTP connection that was established with the server earlier.

2.4 Tools for Producing Mobile Applications

This section provides a brief description of various mobile devices and mobile applications. The project can be developed in various mobile devices such as the iPhone, Windows Phone, and Android Phone. Creating applications on these devices means learning new languages, tools, designs, and procedures. Another method for creating the project could have been a mobile device that is a non-native based HTML application, which is explained later in this chapter. There could have also been different types of communications to the server and different methods of storing data to a database. A more secure way of communicating to the server could have been utilized instead of a non-secure method. Also, there are other methods that could have been performed instead of creating a native app (an application developed for a specific platform) for the project.

Smartphone and Application

A smartphone device is a small handheld device that consists of a touch screen and weighs less than 2 pounds [50]. These devices have Internet, GPS and Near Field Communication (NFC) built into the device. Apps assist users in accomplishing certain tasks on their smartphone devices [36]. There are custom apps that operate with the devices' hardware for completing certain tasks on behalf of the user. Many devices can be used to make phone calls, send and receive text messages, check emails, stay organized, and so on. Apps can be used for such things as: entertainment, business, and searching the web [52]. Many apps permit a user to manipulate the information on the server-side wherever they are, provided that there is an Internet connection. Developers can design an app geared to user requirements and hardware devices.

iPhone

In order to get a valid signature to develop an app and submit applications to the App Store, an Apple Developer must register at the developer website [9]. The tools and languages listed below are essential for developers to design an app for the iPhone device.

MacBook Pro [7] supports the use of a Unix-based operating system [49] known as Mac OS X, which is written in Objective-C that also gives developers an opportunity to create iPhone applications. iOS [3] is the mobile operating system for various Apple devices [47]. The iPhone environment uses the iPhone Application (IPA) to archive the iOS application files. Xcode [10] is used to create the iPhone application requiring an iOS [54] Software Development Kit (SDK). An iOS simulator is used in conjunction with Xcode, which is a helpful tool for developers to simulate the app as if the application were running on an iPhone device [5].

Objective-C is the language that applications are written in which is a set of mostly Smalltalk extensions, that are added to the widely used programming language C. Objective-C gives C full object-oriented capabilities that are simple and straightforward [8]. Cocoa Touch (written in Objective-C [2]) is an Application Programming Interface (API) that is added to the iPhone environment, allows the device to read the information from the user through the hardware [38].

Figure 2.2 iPhone 3GS

There is a device for which the application has been created, that reporters can utilize in the field. The iPhone [6], as shown in Figure 2.2 [46], is an Internet and multimedia-enabled smartphone that runs various services to complete tasks [45].

Windows Phone

In order to be able to develop an app and upload to the App Hub, a Windows Phone Developer must register at the developer website [24]. The tools and languages listed below are essential for developers to design an app for the Windows Phone.

Windows Phone is the mobile operating system for various Windows Phones [42]. Windows is an NT-based operating system [53], which is written in C, C++, and Assembly language, and supports the use of C#. Windows Phone environment uses the Silverlight Application Package (XAP) [12] to archive the Windows application files. Windows Phone Software Development Kit (SDK) is a tool utilized in producing software for Windows Phones [23] which is written in C# [56]. Visual Studios [10] is used to create Windows Phone application, requiring the Windows Phone SDK in order to create applications [43]. Windows Phone simulator is used in conjunction with Visual Studios, which is a helpful tool for developers to simulate the app as if the application were running on a Windows Phone device [23].

C# is a programming language that utilizes object-oriented programming [37]. C# was created for Microsoft's .NET Framework. In creating the design and style of a Windows Phone application [22], a developer needs to know Extensible Application Markup Language (XAML). XAML also performs tasks that users can interact with through the Windows Phone.

Figure 2.3 Windows Phone

The Windows Phone, as shown in Figure 2.3 [21] is a suitable candidate for creating the project as it has advanced features similar to the iPhone. The down side is that the device is not being adopted as fast as the iPhone or Android, which makes it more difficult to achieve a larger user base.

Android Phone

In order to authenticate and upload to Google play, an Android Developer must register at the developer website [15]. The tools and languages listed below are essential for developers to design an app for an Android Phone.

Android is the mobile operating system for various Android devices [34]. The Android environment uses the Android Package Kit (APK) (written in Java [35]) to archive the Android application files. Eclipse is used to create Android Phone applications [13] requiring the Android Software Development Kit (SDK). Eclipse is a tool utilized in producing software for Android [34] devices. The Android simulator is used in conjunction with Eclipse, which is a helpful tool for developers to simulate the app as if the application were running on an Android Phone device.

Java is a programming language that utilizes object-oriented programming [48]. XML is used to design the visual structure of the Android Phone UI and to make debugging problems easier to remedy [17].

Figure 2.4 Androids Phone

The Android Phone, as shown in Figure 2.4 [14] is a suitable candidate for creating the project. Android Phones have far more features that permit the user to accomplish more tasks with the device than the iPhone or Windows Phone. The Android Phone is being adopted at a faster rate because it is cheaper and accessible to all carriers. The down side for the device is that it is being created by numerous companies, which makes it harder for developers to create one app for multiple devices.

Non-Native App

An alternative method for creating a native app on the mobile device is to create an HTML app similar to what was created for this project. An HTML version is accessible to everyone that has an Internet connection and a mobile browser to access it. The HTML app obtains user information the same way as it would for a native application. Developing an HTML app would provide an overall enjoyable user experience because the developer can customize and perform anything without restrictions. The problem with developing an HTML app is that reporters can only use the app when there is an Internet connection. This project needs to be accessed anywhere there are roads, meaning mountains, back roads, and so on. This means that cellular connections may not be obtainable for reporters to upload their observations.

Structured Response System from Server-side to Client-side

XML is a human–readable language that machines understand as well [55]. XML [32] assists the app in validating user input by reading messages from the server. JavaScript Object Notation (JSON) is an easy language for humans to read and write and for machines to understand and parse [18]. JSON is becoming a standard for developers who want to use XML. JSON acts and behaves like XML for storing and exchanging text information [33]. JSON is lighter, faster and easier to parse information from strings.

XML was used instead of JSON, as JSON was not integrated into the developer library until iOS 5.0 [4].

Database Management Systems

SQLite is a public domain that implements most of the SQL standards and is an ACID-Compliant embedded relational database management system [51]. SQLite [28] is the local database used on the mobile device for storing report information.

Derby is an open source relational database implemented in Java [1]. Derby has a small footprint, is SQL standardized, and operates directly from disk. Derby is written in Java, which supports the Java VM that accesses the database. Derby utilizes more memory occupying several megabytes compared to SQLite, which occupies less than a few hundred kilobytes [30].

Dbstar (Db.*) reads and writes to a database utilizing the libraries from C/C++, implying that only C/C++ programs can utilize the database. Db.* utilizes less memory, occupying fewer kilobytes than SQLite. SQLite binds to various programming languages meaning that any program can utilize the database [29].

Firebird is an embedded database system that can be developed to a full size enterprise-class client/server database. SQLite is an embedded database system only. Firebird utilizes more memory occupying several megabytes compared to SQLite, which occupies less than a few hundred kilobytes [31].

Communication Protocol between Client-side to Server-side

Hypertext Transfer Protocol Secure (HTTPS) is a procedure that encrypts the information that is being transferred between client-side and server-side. HTTPS guarantees authenticity, confidentiality, and integrity from client to server. This type of approach prevents others from eavesdropping and man-in-the-middle attacks [44]. This project can benefit from HTTPS as the information being collected is considered to be sensitive data. In order to obtain a valid HTTPS communication, the developer must buy an SSL Certificate, which can range from several US dollars to thousands of USD. There is a free SSL Certificate that can be used, but that method has limitations that users may not deem secure enough. An HTTPS method was not used in this project, as security was not taken into account when the project was initially created.

Alternative Mobile Devices

There are other mobile devices that can be utilized in permitting a reporter to upload their observations such as: laptops, netbooks, and tablets. Laptops have larger screen sizes and can accomplish additional tasks that certain mobile devices cannot. Laptops are not ideal for this project because reporters are not keen about lugging around a heavy laptop with no cellular access to report road kill observations. Netbooks are smaller and lighter, but are less resource intense than laptops. They are also not ideal because reporters need more resources to do various other tasks. Tablets contain larger screen sizes and have similar features to the various smartphones that are being utilized daily. However, tablets are not ideal, either, because of their higher cost, which limits accessibility.

Alternative Methods for Data Collection

There are various ways of collecting wildlife-vehicle collision data, which could have been utilized. A mass email could have been sent to various people asking for information about their observations. This method may be considered as spam for most people and therefore not the best choice. A phone call to various people could have been made asking for information about any observations on road kill. Most people do not like this method as they are busy and consider telemarketers annoying. A survey could have been made asking people for observations while they exit stores. This method would not yield enough data, as many individuals are not interested in taking surveys while in a rush to get home. Various individuals could complete a questionnaire about their observations. This method would not yield enough data either, as individuals are not interested in answering questions throughout their busy day.

2.5 Similar Apps that Address Environmental Issues

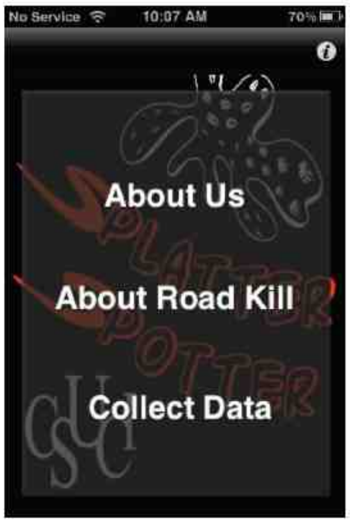


Figure 2.5 Roadkill App

The Roadkill app shown in Figure 2.5 [11] is similar to what is being developed for this project. The app accepts road kill observations from commuters. The commuters can complete a single report by dropping a pin onto a map and filling out the necessary information about the road kill. Also, a transect feature permits the commuters to take multiple reports while observing a lengthy stretch of road. The report is then sent to a data analyzer's email account for further processing.

Figure 2.6 Street Bump App

Street Bump, as shown in Figure 2.6 [25], is an app that detects potholes and reports them to city officials. As of this write up, the app is still being developed, but has some features that are worth noting. Commuters using this app are able to record “bumps” or potholes with the use of the iPhones’ accelerometer. Once the device has detected a pothole, a GPS location is retrieved and the report is uploaded to the server for later analysis. The app provides city officials with reports of potential road problems.

Chapter 3: Functions Utilized for the Mobile Application

This chapter discusses a variety of functions that are utilized in the mobile application that was created. An individual can set default units of measurement based on user preferences. The mobile app allows for such mechanisms as single, transect, and incomplete reports. The app can store reports remotely on the server as a single report or as multiple reports. A user has the ability to store reports, view completed reports, and omit reports that are stored locally on the application. The app has a notification system that reminds an individual of how many incomplete reports remain.

3.1 Different ways of Recording a Report

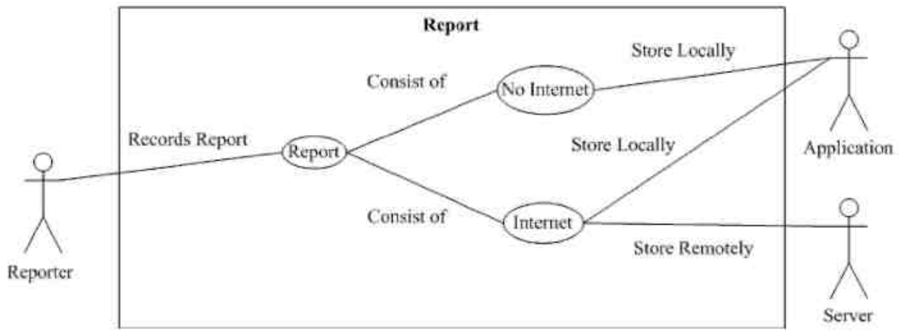


Figure 3.1 Different ways of Recording a Report

A reporter can record a report, which consists of Internet or No Internet, as shown in Figure 3.1. No Internet reports are stored locally on the application and can later be stored remotely with an Internet connection. Internet equipped reports are stored locally on the application and stored remotely on the server.

3.2 Reporting with Transect

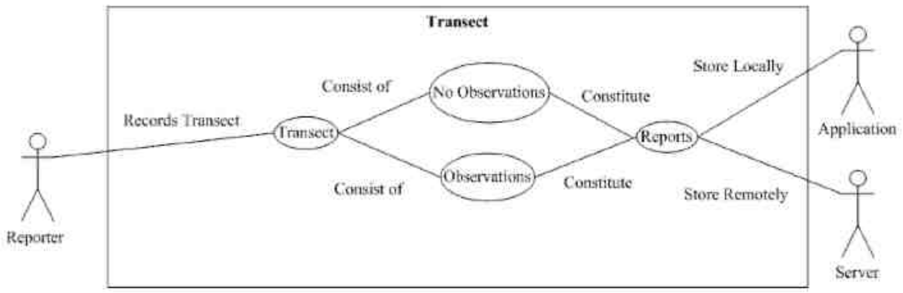


Figure 3.2 Reporting with Transect

A reporter can record a transect that consist of "observations" or "no observations". "No observation" is classified as a report because this information is important to document as well. Observations are collections of wildlife-vehicle collision data, which constitute a report. These reports are then stored locally on the application and stored remotely on the server, as shown in Figure 3.2. A transect is stored locally to the application if there is no Internet connectivity. Transect reports can be stored to the remote database when there is an Internet connection.

3.3 Submitting a Report

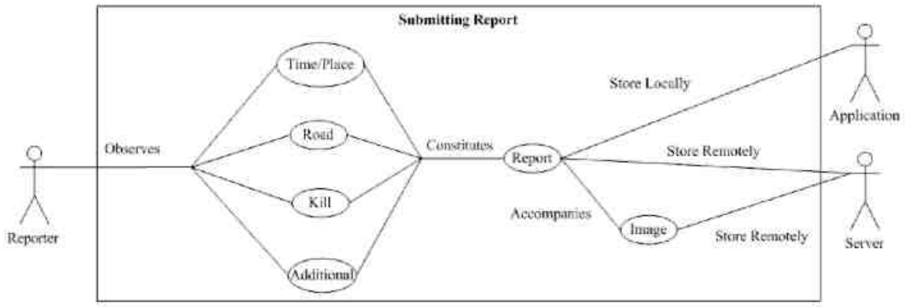


Figure 3.3 Submitting Report

A reporter has to record time/place, road, kill, and additional data about the wildlife-vehicle collision before storing it on the server. This information constitutes a report, as shown in Figure 3.3. Any report taken by the reporter is stored locally on the application for later review. If a report is accompanied by an image, then the report is stored to the remote server with a value in the image parameter. If a report does not have an image, then the value of the image parameter is empty. The reporter can submit reports to the remote server when the mobile device has an Internet connection.

3.4 Uploading Multiple Reports

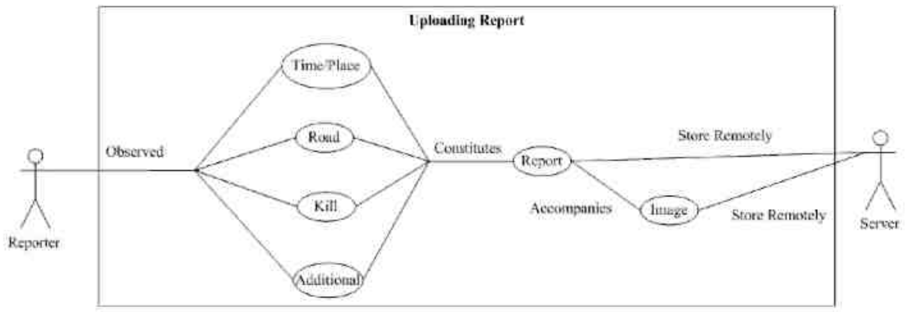


Figure 3.4 Uploading Multiple Reports

A reporter can upload multiple reports to the remote server. Each report constitutes an observation containing time/place, road, kill, and additional, as shown in Figure 3.4. To successfully upload all reports, each field needs to be completely filled in. If a report does not have an image, then the reporter can still store the report remotely to the server, as an image is optional. If a report is accompanied by an image, then the report is stored to the remote server with a value in the image parameter. If there is more than one incomplete report, then the app reiterates the process for all incomplete reports. The reporter can submit reports to the remote server when the mobile device has an Internet connection.

3.5 Omitting Items from the Database

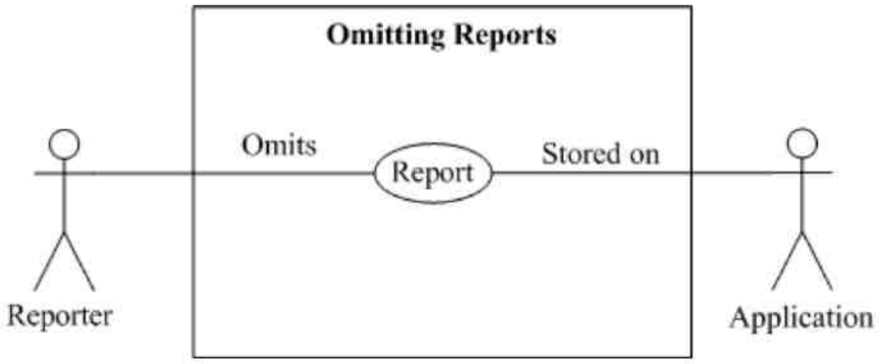


Figure 3.5 Omitting Items from the Database

Omitting items is useful as reports can grow exponentially. Omitting consists of deleting or purging reports from the application, as shown in Figure 3.5. The reporter can omit all reports, selected reports, or completed reports. A report can be complete and/or incomplete.

3.6 Storing Reports Locally

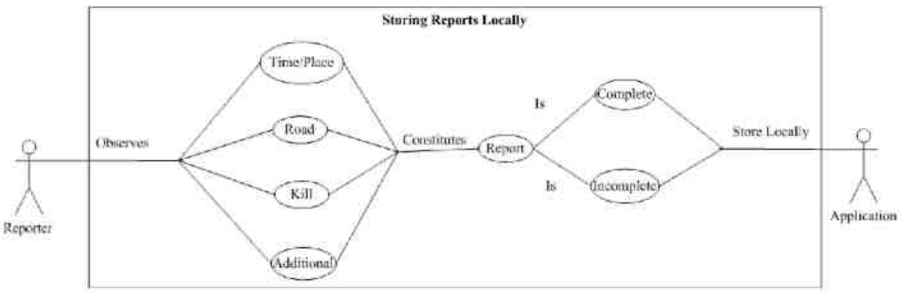


Figure 3.6 Storing Reports Locally

A reporter can record some or all of time/place, road, kill, and additional data for a specific observation. The different categories constitute a report and are ready to be marked as complete or incomplete, as shown in Figure 3.6. If a report has not been thoroughly completed, then the report is marked as incomplete. Reports that are submitted remotely to the server are marked as complete. Complete and incomplete reports are stored locally.

3.7 Resuming Incomplete Reports

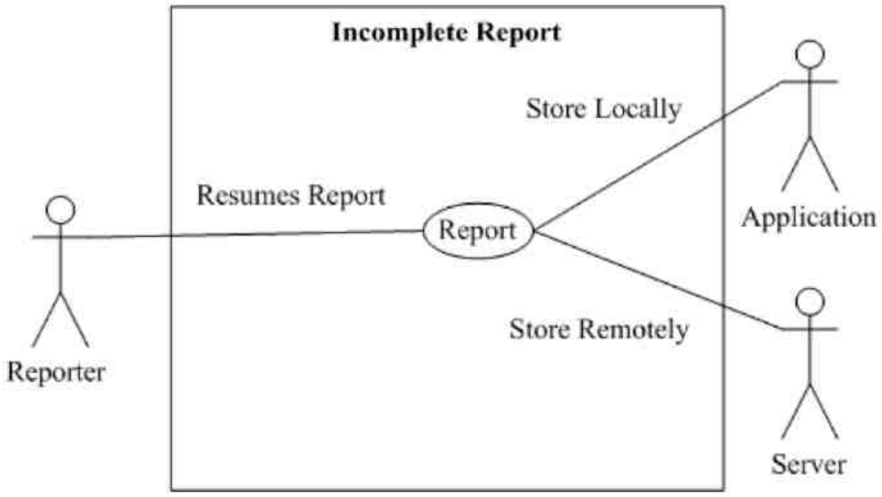


Figure 3.7 Resuming Incomplete Reports

A reporter can resume any report that has not been stored remotely to the server. A reporter is able to store a report locally and remotely after modifications have been made to the single report, as shown in Figure 3.7. Incomplete reports are stored locally to the application if there is no Internet connectivity or if the report is still incomplete. When there is an Internet connection, a reporter can store incomplete reports to a remote database.

3.8 Displaying Completed Reports

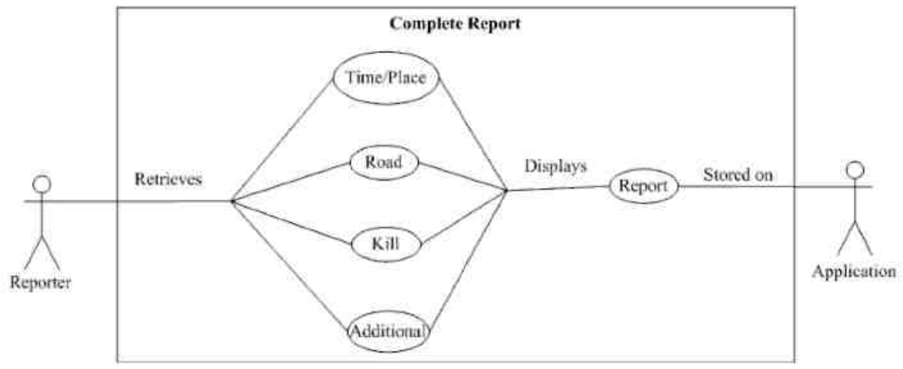


Figure 3.8 Displaying Completed Reports

A reporter can view previously submitted reports of past observations. This feature allows the reporter to keep a log of all reports that they have completed. A completed report is comprised of time/place, road, kill, and additional data, as shown in Figure 3.8.

3.9 Badge Notification System

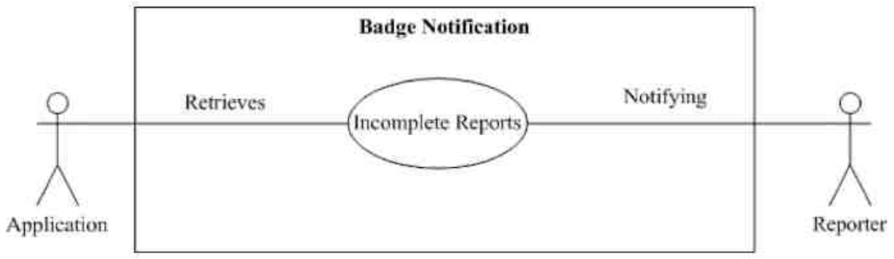


Figure 3.9 Badge Notification System

A notification system such as a badge number is useful in reminding reporters that there are incomplete reports that need to be completed. The application retrieves all incomplete reports stored within the local database and updates the badge number accordingly, as shown in Figure 3.9.

3.10 Setting Default Units of Measurement

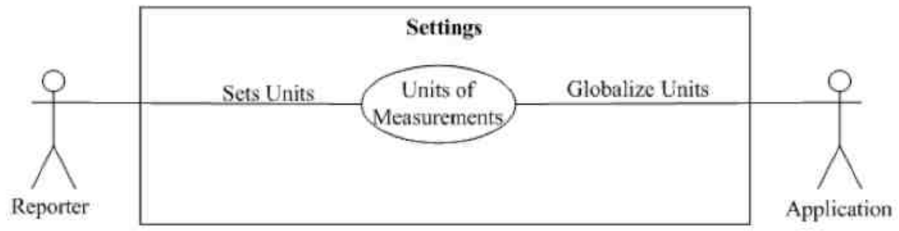


Figure 3.10 Setting Default Units of Measurement

A reporter is capable of adjusting default units through settings, as shown in Figure 3.10. The units are globally changed throughout the application after the adjustments have been made. Units of measurement are Imperial units or International system of units.

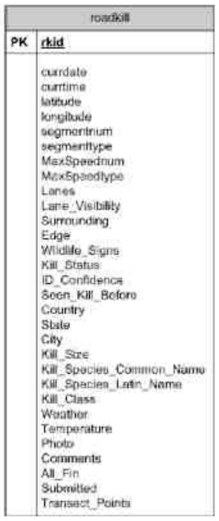
Chapter 4: In-depth Description of the Design and Functionalities of the Application

This chapter gives an in-depth description of the design and functionalities of the application. There were various tools that were employed for creating the application. A visual display of the database helps in explaining the table and attributes that were used for this project. The application permits setting default units of measurements. A reporter has the ability to generate a single or transect report. Uploading a single report from the mobile device to the server is explained. The app provides a method for adjusting incomplete reports and reviewing completed reports stored locally. A method is given for uploading incomplete observations to the remote database. A reporter can omit reports from the local database on the device without modifying the remote database. A notification system was implemented for alerting a reporter about an incomplete report.

4.1 Mobile Device Tools Utilized

iOS is the operating system that was chosen, which allows an application to be used on the mobile device. Xcode is the tool used for developing an application for the mobile device. Objective-C is the language used to construct the mobile application. iOS Simulator is used to test the application through a virtual environment. Cocoa Touch is the language used to manipulate the application with various finger gestures. The iPhone allows for finger gestures and access to mobile device hardware features. Mac OS X is the operating system that was chosen, which allows for creating mobile device applications for the iPhone. SQLite is a database management system for executing queries, storing data, and retrieving data. XML is the language used as a response from the server-side to the client-side with information to manipulate the application.

4.2 Conceptual Database Design



roadkill	
PK	rkid
	currdate
	currtime
	latitude
	longitude
	segmentnum
	segmenttype
	MaxSpeednum
	MaxSpeedtype
	Lanes
	Lane_Visibility
	Surrounding
	Edge
	Wildlife_Signs
	Kill_Status
	ID_Confidence
	Seen_Kill_Before
	Country
	State
	City
	Kill_Size
	Kill_Species_Common_Name
	Kill_Species_Latin_Name
	Kill_Class
	Weather
	Temperature
	Photo
	Comments
	All_Fin
	Submitted
	Transect_Points

Figure 4.1 Conceptual Database Design

The database table is entitled Roadkill, which consists of attributes that are essential for a detailed report, as shown in Figure 4.1. The attributes are: rkid, currdate, currtime, latitude, longitude, segmentnum, segmenttype, MaxSpeednum, MaxSpeedtype, Lanes, Lane_Visibility, Surrounding, Edge, Wildlife_Signs, Kill_Status, ID_Confidence, Seen_Kill_Before, Country, State, City, Kill_Size, Kill_Species_Common_Name, Kill_Species_Latin_Name, Kill_Class, Weather, Temperature, Photo, Comments, All_Fin, Submitted, and Transect_Points.

The SQL database requires data types to determine what values are going to be inserted into the database (Appendix A, A.1). Data types are useful because the app needs to know how to retrieve those values and display them in the specific fields accordingly. The following data types are used: integer, tinyint, text, varchar, and blob.

Data Types

INTEGER

Signed integer can be stored as 1-4, 6, or 8 bytes. TINYINT is considered to be an integer as well.

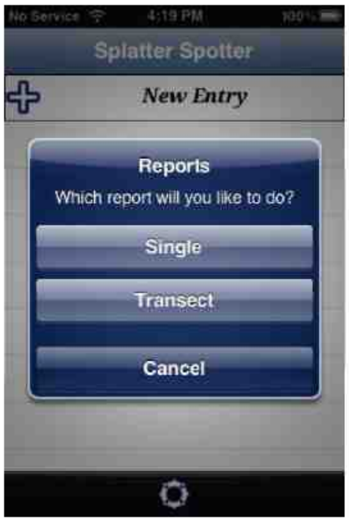
TEXT

Text string can be encoded in UTF-8 and stored into the database. VARCHAR is considered to be text as well.

BLOB

Blob stores the data exactly as it is received.

4.3 Initiating a Report with New Entry



Reports

Which report will you like to do?

Figure 4.2 Initiating a Report with New Entry

The "New Entry" button initializes a report selection, as shown in Figure 4.2. An alert message pops up letting the user choose from a single report or transect report. Based on the device's Wi-Fi/3G capabilities, a user is taken to one of two different report views for a single report. If Wi-Fi/3G is enabled, then the user is taken to a report that auto fills some of the fields. If Wi-Fi/3G is not enabled, then the user is taken to a report where each field needs to be manually completed. If a transect report is selected, then the user is taken to the transect view.

4.3.1 Single Report

There are four diverse categories in the single report: Time/Place, Road, Kill, and Additional.

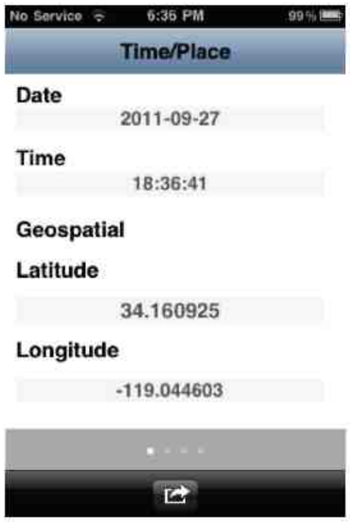
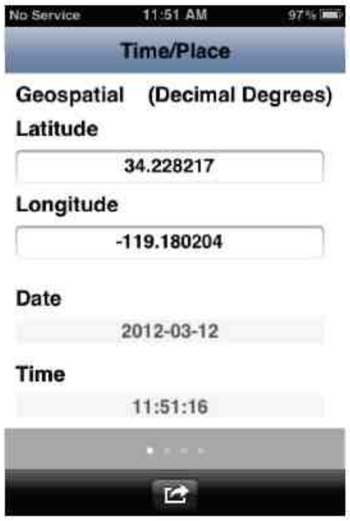


Figure 4.3 Automatically Retrieving Time/Place

The Time and Place category takes the user's date, time, latitude, and longitude at the point of initializing the report, as shown in Figure 4.3.

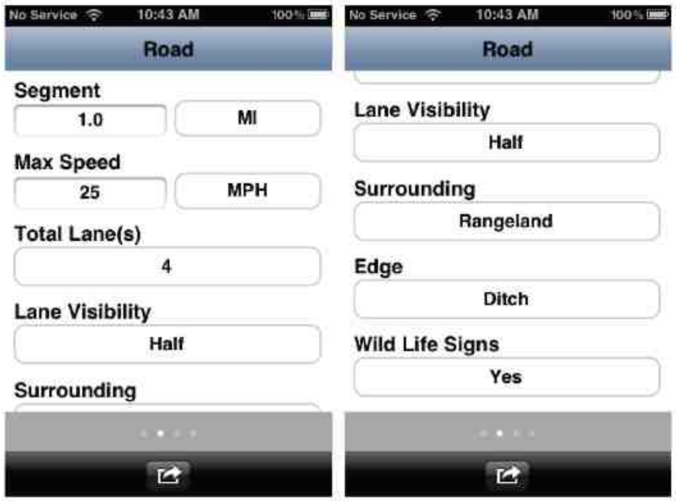
Information from the mobile device is automatically inserted into its respective areas with specific information. Date and time are retrieved from the mobile device itself and formatted to present time as HH:mm:ss and date as yyyy-MM-dd With an Internet connection, the latitude and longitude are automatically collected and inserted into their fields as decimal degrees from the location manager library. Without Internet connection, the latitude and longitude attempts to automatically input the location of the device as decimal degrees from the location manager library.



Geospatial (Decimal Degrees)
Latitude
34.228217
Longitude
-119.180204
Date
2012-03-12
Time
11:51:16

Figure 4.4 Manually Inserting Place

If the location is not acquired for whatever reason, then the user has to manually insert the latitude and longitude of the report in decimal degrees, as shown in Figure 4.4. The placeholder displays how to input the information into the text field. After the information has been inserted into the text field, the app validates and notifies the user if the information is invalid. Regular expressions are used to validate both latitude and longitude text fields. The regular expression verifies if the string is between -90.000000 and 90.000000 for latitude and -180.000000 and 180.000000 for longitude. If the input is still invalid, then an exclamation mark error button is displayed next to the invalid text fields. Once the button is pressed, an alert message displays a valid format for the text field. After the user has inserted valid information into the text field and presses the done button, the numbers and punctuation keyboard is dismissed.


No Service 10:43 AM 100%

Road

Segment
1.0 MI

Max Speed
25 MPH

Total Lane(s)
4

Lane Visibility
Half

Surrounding

No Service 10:43 AM 100%

Road

Lane Visibility
Half

Surrounding
Rangeland

Edge
Ditch

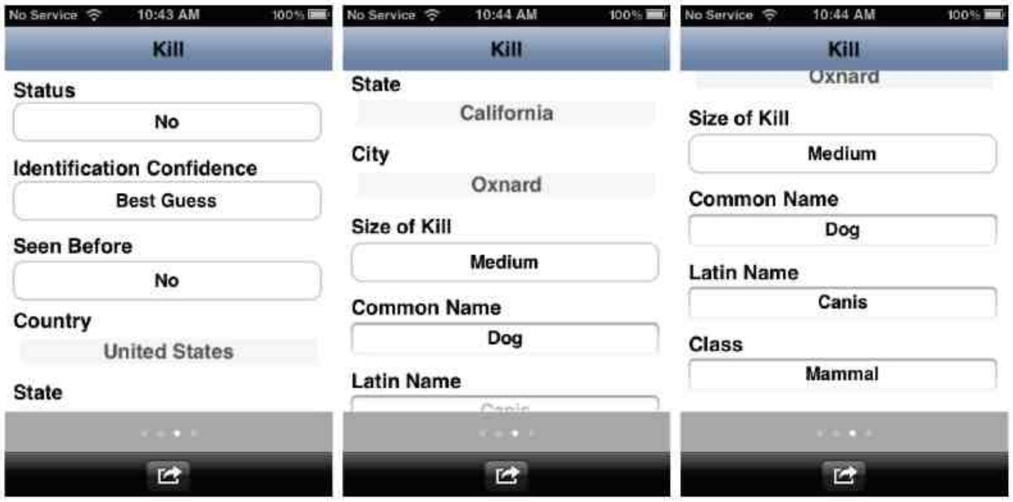
Wild Life Signs
Yes

Figure 4.5 Inserting Road Information

The Road category takes the user's information about the road upon which the road kill was found, as shown in Figure 4.5.

The text fields display a number and punctuation keyboard. The user is able to input a whole number or a decimal number to the tenth place. The placeholder displays how to input the information into the text field. After the information has been inserted into the text field, the app validates and provides notification if the information is invalid. Regular expressions are used to validate both segment and max speed text fields. The regular expression verifies if the string is either a whole number or a decimal number to the tenth place. If the input is still invalid, then an exclamation mark error button is displayed next to the invalid text fields. Once the button is pressed, an alert message displays a valid format for the text field. After the user has inserted valid information into the text field and presses the done button, the numbers and punctuation keyboard is dismissed.

Based on the unit of measurements selected in the settings page, the segment and max speed pickers are predetermined to a unit standard. Every picker has its own set of values. Users can adjust any picker to provide precise information about the road. Once the user has selected a value for the picker, the title changes and the position of the value is retained. If the user wants to adjust the value to something else, then the picker scrolls to the value previously chosen and allows the user to change to a different value.



Status
No
Identification Confidence
Best Guess
Seen Before
No
Country
United States
State

State
California
City
Oxnard
Size of Kill
Medium
Common Name
Dog
Latin Name
Canis

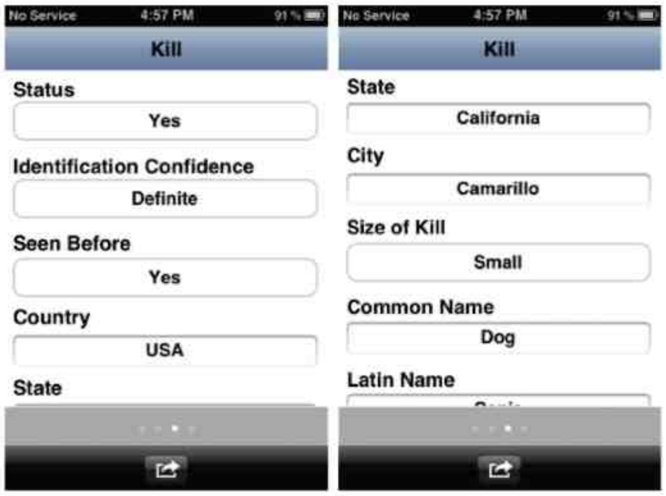
Oxnard
Size of Kill
Medium
Common Name
Dog
Latin Name
Canis
Class
Mammal

Figure 4.6 Inserting Killed Information

The Kill category takes in user information about the dead animal that was found, as shown in Figure 4.6.

Information from the mobile device is automatically inserted into their respective areas with specific information. With an Internet connection the country, state, and city are automatically inserted from the reverse geocoder library. A reverse geocoder takes the latitude and longitude previously retrieved and establishes the place mark of the device. The place marks used in this app are country, locality, and administrativeArea. All remaining fields have to be manually inserted. The placeholder displays how to input the information into the text field. After the information has been inserted into the text field, the app validates and notifies the user if the information is invalid. Regular expressions are used to validate all text fields. The regular expression verifies if the string contains upper/lower alphabet characters with or without spaces to a maximum of 255 characters. If the input is still invalid, then an exclamation mark error button is displayed next to the invalid text fields. Once the button is pressed, an alert message displays a valid format for the text field. After the user has inserted valid information into the text field and presses the done button, the alphabetical keyboard is dismissed.

Users can adjust any picker to provide precise information about the kill. Once the user has selected a value for the picker, the title changes and the position of the value is retained. If the user wants to adjust the value to something else, then the picker scrolls to the value previously chosen and allows the user to change to a different value.

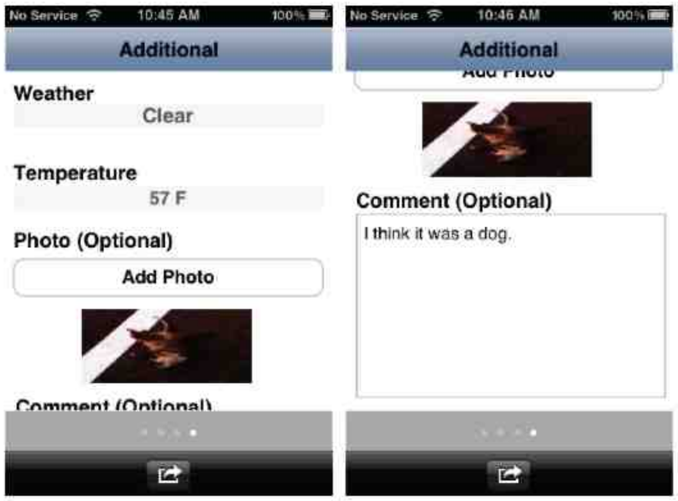


Status
Yes
Identification Confidence
Definite
Seen Before
Yes
Country
USA
State

State
California
City
Camarillo
Size of Kill
Small
Common Name
Dog
Latin Name

Figure 4.7 Manually Inserting Kill

Without an Internet connection, the user has to manually insert country, state, and city using an alphabetical keyboard for the kill, as shown in Figure 4.7. All remaining fields have to be manually inserted with the previously mentioned methods from the automatic insertion.



Weather
Clear
Temperature
57 F
Photo (Optional)
Add Photo
Comment (Optional)

Add Photo
Comment (Optional)
I think it was a dog.

Figure 4.8 Supplementary Information Insertions

The Additional category takes in additional information about the dead animal, as shown in Figure 4.8.

With an Internet connection, the weather and temperature are automatically inserted into the weather and temperature fields. The latitude and longitude that were previously obtained from the mobile device are used in acquiring the weather through an online weather API [16]. Weather information is posted back to the app through HTTP as

XML. To acquire the current weather and temperature values, the XML string is parsed using the NSXMLParser.

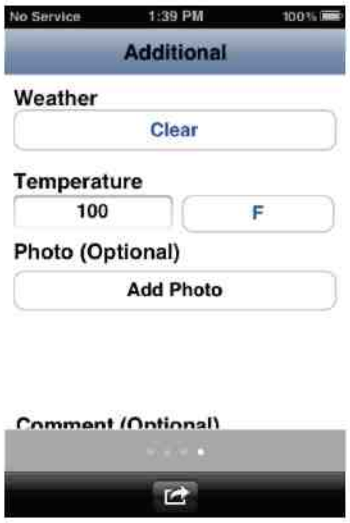


Figure 4.9 Manually Inserting Supplementary Information

Without an Internet connection, the user has to manually select weather and the temperature unit, as shown in Figure 4.9. Weather is defaulted to clear. Based on the unit of measurements selected in the settings page, the temperature picker is predetermined to a unit standard. Users can adjust pickers to provide precise information about weather and temperature. Once the user has selected a value for the picker, the title changes and the position of the value is retained. If the user wants to adjust the value to something else, then the picker scrolls to the value previously chosen and allows the user to change to a different value.

Without an Internet connection, the user has to manually input the temperature into the text field, as shown in Figure 4.9. The placeholder displays how to input the information into the text field. After the information has been inserted into the text field, the app validates and notifies the user if the information is invalid. A regular expression is used to validate the temperature text field. The regular expression verifies if the string contains a whole number with a maximum of three numbers. If the input is still invalid, then an exclamation mark error button is displayed next to the invalid text fields. Once the button is pressed, an alert message displays a valid format for the text field. After the user has inserted valid information into the text field and presses the done button, the numbers and punctuation keyboard is dismissed.

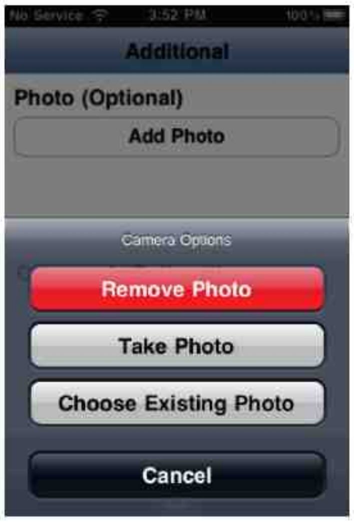


Figure 4.10 Actions for Adding and Removing Photo

An optional photo can be selected if a user presses the "Add Photo" button, as shown in Figure 4.10. Doing so permits the user to select from three different actions: remove photo, take photo, and choose existing photo. "Remove Photo" assures that the image view is set to nil and the image is no longer displayed. "Take Photo" calls the image picker controller class, which permits the user to utilize the built-in camera on the mobile device and places the image to the image view. "Choose Existing Photo" calls an image picker controller class, which permits the user to select a photo from the camera roll or photo library on the mobile device and places the image to the image view. After an image has been selected, the user is able to tap on a thumbnail of the image to display a full sized image.

A text view is then displayed to let the user input optional comments if they believe more information is required for the report. When the text view is activated, a done button is located on the upper right of the additional category. This button assists in dismissing the keyboard when done. After the information has been inserted into the text view, the app validates and notifies if the information is invalid. A regular expression is used to validate the text view comments. The regular expression verifies if the string contains alphabet, numbers, periods, and spaces with a maximum size of 255 characters. If the input is still invalid, then an exclamation mark error button is displayed next to the invalid text view. Once the button is pressed, an alert message displays a valid format for the text view. After the user has inserted valid information into the text view and presses the done button, the keyboard is dismissed.

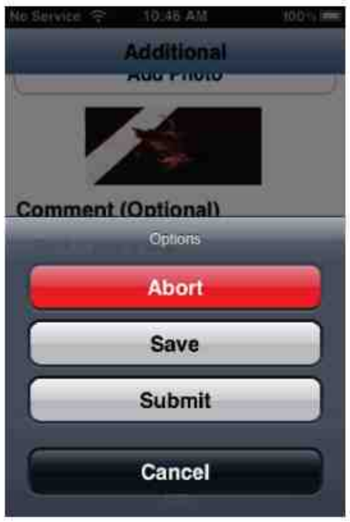


Figure 4.11 Actions for Processing a Report

Pressing the "Action" button allows for processing the report with three alternative choices: "Abort", "Save", or "Submit", as shown in Figure 4.11.

"Abort" relocates to the home page without saving and submitting the current report.

The app saves the current report locally. To save the report correctly, there are preventions implemented. A function is called before the data is saved, which scrutinizes latitude, longitude, and temperature against regular expressions mentioned earlier for invalid field entry. The prevention is implemented, so that the data being requested from the Internet has time to be retrieved and stored in certain fields. To save to the local database, an SQL statement is required. An insert statement is executed to collect all variables from the four different categories mentioned. The values from a reporter are inserted into the database to their corresponding attributes listed in the Roadkill database. As the information is being saved to the local database, an activity indicator is displayed informing the user that the app is still processing the information. Also, the app determines if all fields are filled in 1 or not filled in 0 and sets the submitted attribute value to 0. Once the app has been successfully inserted into the local database, the app relocates to the home page.

If there is Internet connectivity, the "Submit" button is enabled to submit the report. First, the program verifies if all required fields are filled in. If it is not filled in correctly, then the user is alerted that there are field values missing before they are able to submit. Second, the program verifies if there is Internet by asking for an Internet page request. Lastly, the app verifies if the server is active and the upload file located on the server is accessible. If either Internet or server is not obtainable, then an alert from the app notifies the user about the problem and the data is saved locally as mentioned above.

If the fields are filled in correctly and Internet/server is obtainable, then the program proceeds to send the data to the server. First, if there is an image, the image is uploaded to the server as a 'jpg file. The app sets up a URL request with an HTTP method of POST. A unique valid HTML post is created with a generated random number for the header. A content type for the HTTP header field of multipart/form-data is set. The body of the post is created with a start header, the name of the image, content type of image/jpeg, image data being used, and ending header. Every step of the body except the image data being used is encoded using the UTF-8 encoding system. A URL connection is made with a synchronous request of the body and a response from the server. The server replies with an XML response containing one of the following: an image file name, image file error, or empty file.

Second, a new post is created containing all the information from the report and the response from the server. The string is encoded in ASCII and the length of the post being sent is obtained. The app sets up the URL with the string of information inserted from the report. The HTTP method is set to POST. The header is started with length of the post that was encoded earlier and the content-type is set to application/x-www-form-urlencode. A URL connection is made again with a synchronous request of the new post and a response from the server. The server replies with an XML response containing one of the following: submit error or submit success. If submit error is returned, then the app parses the corresponding error messages from the server. The XML tags given are flagged in the app as errors for the specific fields that need to be changed. If submit success is received, then the app saves locally. As the information is being uploaded to the server and updated to the local database, an activity indicator is displayed informing the user that the app is still processing the information. As the information is being saved and uploaded to the server, the app sets all fields as filled in with the value of 1 and submitted with the value of 1.

4.3.2 Mechanisms of Transect

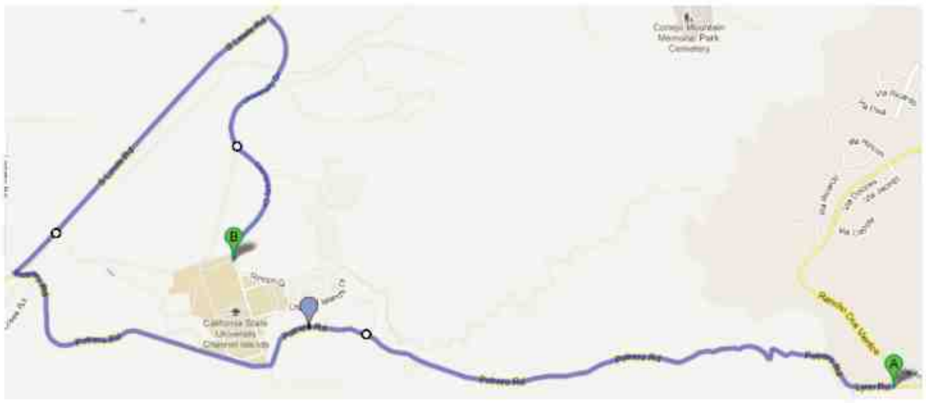

B
A
California State University Channel Islands
Potrero Rd
Conejo Mountain Memorial Park Cemetery

Figure 4.12 An Example of Transect with Waypoints

A transect is a different type of report consisting of one or more reports that includes waypoints, as shown in Figure 4.12.

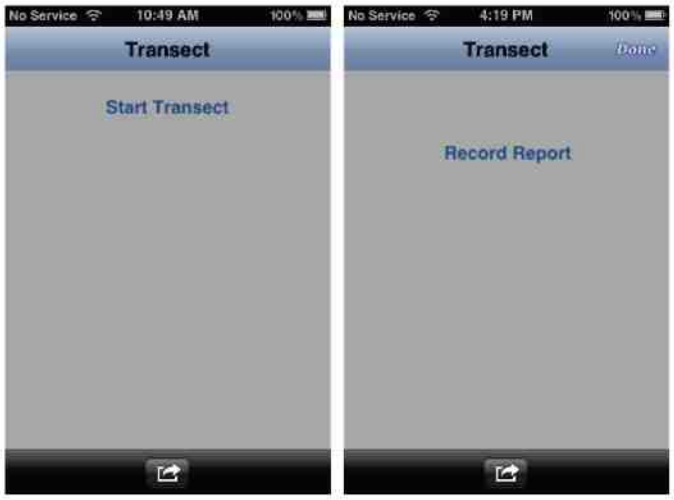


Figure 4.13 Mechanisms of Transect

After transect is initiated, a reporter is presented with three buttons, as shown in Figure 4.13. The first button, “Start Transect”, permits a reporter to initiate a transect report, which obtains the origin location and initializes the waypoints to be called every five minutes. The second button, “Record Report”, lets a reporter record a report while in transect mode. The reporter can record data utilizing the single report method. Actions for this report are modified to “Abort” and “Save”. Abort cancels the current report and relocates to the transect page. Save stores the report into the local database as a single report and adds a 1 to the Transect_Points attribute. The third button, “Done” stops the waypoints function from being called and obtains the destination location. An array containing Transect_Points equaling 1 is created. Using this array, an SQL update statement is called to insert the start, way, and end points to the Transect_Points attribute as a single string. This string separates latitude and longitude with commas.

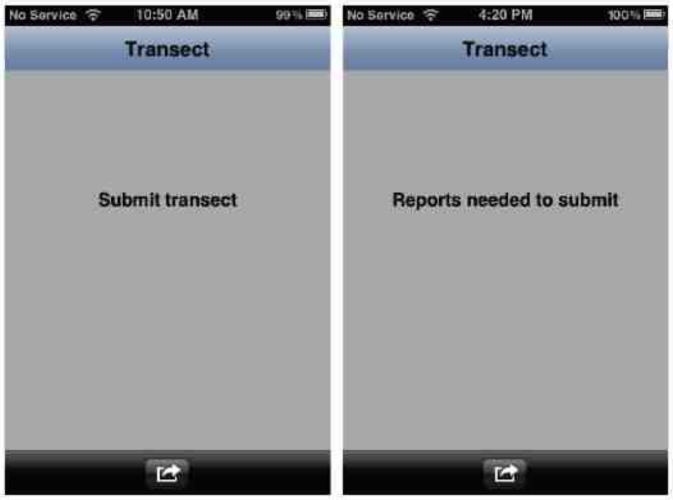


Figure 4.14 Notifying that Action is needed

A label is displayed when the "Done" button is clicked and notifies the reporter that further action is needed, as shown in Figure 4.14. If there are report(s), then the label displays "Reports needed to submit". If there are no reports, then the label displays "Submit transect".

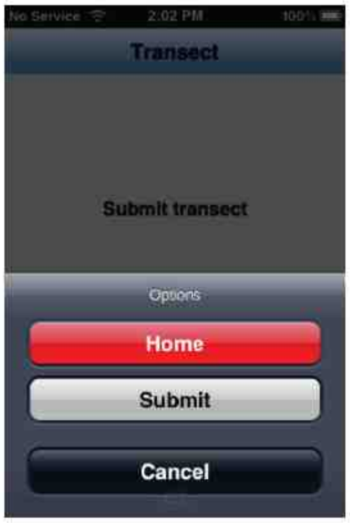


Figure 4.15 Actions for Processing Transect

The "Action" button shown in Figure 4.14, allows the user to process a complete transect, as shown in Figure 4.15.

If the reporter has started the report and does not want to continue, then the home button stops the waypoints from being called and relocates to the home page.

"Submit", allows the user to upload transect information to the server. If Wi-Fi/3G is unobtainable, then submitting saves the report(s) locally and the user is able to submit later. When Wi-Fi/3G is obtainable, then the user is able to upload the report(s) remotely and update locally. To do this, the primary numbers that were stored in the array when the “Done” button was pressed are used again. A for loop is used to process the primary numbers in the array. An SQL select statement is used to select all data from the database using the primary number in the array, which stores these values into temporary variables.

If the fields are filled in correctly and Internet/server is obtainable, then the program proceeds to send the data to the server. First, if there is an image, the image is uploaded to the server as a .jpg file. The app sets up a URL request with an HTTP method of POST. A unique valid HTML post is created with a generated random number for the header. A content type for the HTTP header field of multipart/form-data is set. The body of the post is created with a start header, the name of the image, content type of image/jpeg, image data being used, and ending header. Every step of the body except the image data being used is encoded using the UTF-8 encoding system. A URL connection is made with a synchronous request of the body and a response from the server. The server replies with an XML response containing one of the following: an image file name, image file error, or empty file.

Second, a new post is created containing all the information from the report and the response from the server. The string is encoded in ASCII and the length of the post being sent is obtained. The app sets up the URL with the string of information inserted from the report. The HTTP method is set to POST. The header is started with the length of the post that was encoded earlier and the content-type is set to application/x-www-form-urlencode. A URL connection is made again with a synchronous request of the new post and a response from the server. The server replies with an XML response containing one of the following: submit error or submit success. If submit error is returned, then the app continues to the next primary number in the array. If submit success is received, then the app updates locally. As the information is being uploaded to the server and updated to the local database, an activity indicator is displayed informing the user that the app is still processing the information. As the information is being saved and uploaded to the server, the submitted value is set to 1 by the app. The process is repeated until all primary numbers in the array have been processed. If the report has not been completed, then the upload steps are skipped for that report to the next primary number in the array. Once every report has been processed, the user is relocated to the home page. If there are no reports recorded, then a single report is saved locally and sent to the remote database (same steps above) with default values and transect points.

4.4 Uploading Incomplete Items to the Server

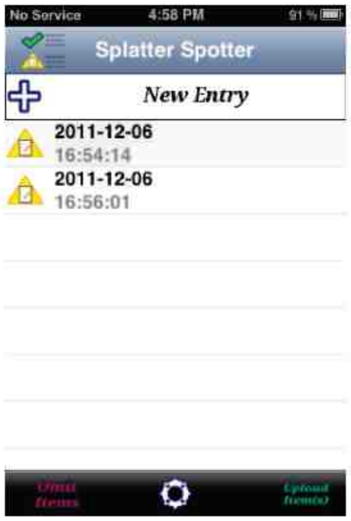


Figure 4.16 Uploading Incomplete Items to the Server

If there are incomplete entries that are ready to be uploaded to the server, then the "Upload Item(s)" button is displayed on the lower right of the home page view, as shown in Figure 4.16. Uploaded item(s) permit the user to upload multiple entries to the server. An SQL select statement retrieves Submitted reports that equal to 0 and All_Fin that equal to 1 from the local database, and stores the primary numbers in an array.

A for loop is used to process the primary numbers in the array. An SQL select statement is used to select all data from the database using the primary number in the array, which stores these values into temporary variables.

After the app verifies for Internet and file accessibility, then the item(s) can be uploaded to the server one at a time. First, if there is an image, the image is uploaded to the server as a .jpg file. The app sets up a URL request with an HTTP method of POST. A unique valid HTML post is created with a generated random number header. A content type for the HTTP header field of multipart/form-data is set. The body of the post is created with a start header, the name of the image, content type of image/jpeg, image data being used, and ending header. Every step of the body except the image data being used is encoded using the UTF-8 encoding system. A URL connection is made with a synchronous request of the body and a response from the server. The server replies with an XML response containing one of the following: an image file name, image file error, or empty file.

Second, a new post is created containing all the information from the report and the response from the server. The string is encoded in ASCII and the length of the post being sent is obtained. The app sets up the URL with the string of information inserted from the report. The HTTP method is set to POST. The header is started with length of the post that was encoded earlier, and the content-type is set to application/x-www-form-urlencode. A URL connection is made again with a synchronous request of the new post and a response from the server. The server replies with an XML response containing one of the following: submit error or submit success. If a submit error is returned, then the app continues to the next primary number in the array. If submit success is received, then the app updates locally. As the information is being uploaded to the server and updated to the local database, an activity indicator is displayed informing the user that the app is still processing the information. As the information is being saved and uploaded to the server, the submitted value is set to 1 by the app. The process is repeated until all primary numbers in the array have been processed. If Internet is not obtainable between entries, then the program breaks from for loop and stops uploading information to the server.

4.5 Omitting Items from the Local Database

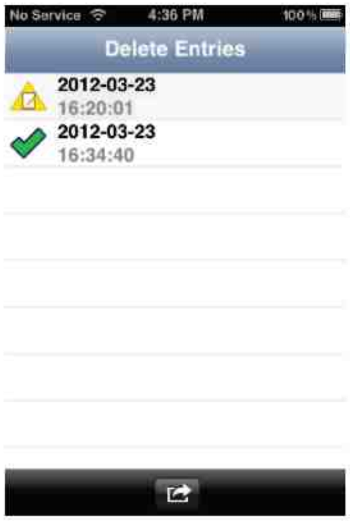


Figure 4.17 Omitting Items from the Local Database

When there are reports in the database, the “Omit Items” button is displayed on the lower left of the home page view, as shown in Figure 4.16. The “Omit Items” button relocates to the Delete Entries view, as shown in Figure 4.17. The table view displays reports from the local database. The table view has a function that counts the number of rows in the database and presents that many entries. This table view retrieves data from the database and presents the date and time in one record cell. The date is displayed as the text label of the report while the time is displayed as the detail text label of the report. A report that has been submitted is accompanied with a green check mark. If the record has not been submitted, then a non-submitted image is shown instead. To delete a selected report from the local database, tap a report to reveal a check mark to the right of the report. The primary number is collected into an array when the report has been selected and waits for further processing. Tapping the report again conceals the check mark and the primary number is removed from the array.

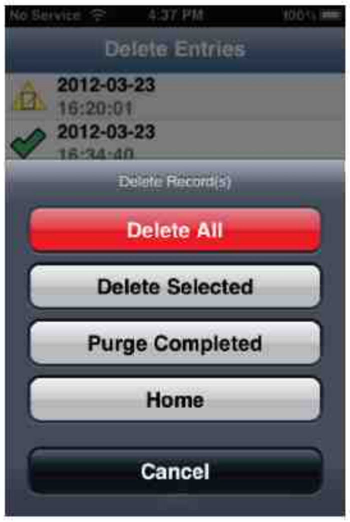


Figure 4.18 Actions for Omitting Reports

The "Action" button allows for deleting all, deleting selected, purging completed, and returning home, as shown in Figure 4.18. Home relocates to the home page.

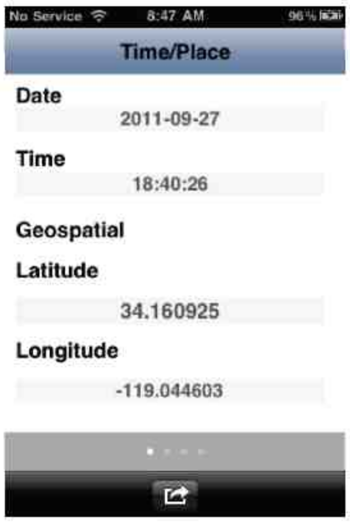
"Delete All" removes all complete and incomplete reports from the local database. Before removing the reports, an alert message is displayed confirming "YES" to delete or "NO" to not delete. If a user has chosen "YES", then an SQL select statement selects all reports in the local database collecting the primary numbers into in array. A for loop is used to delete all primary numbers in the array with an SQL delete statement. After all reports have been processed, the user relocates to the home page.

"Delete Selected" removes the selected reports from the local database. The primary numbers are collected into in array when the user taps on a report as mentioned earlier. A for loop is used to delete all primary numbers in the array with an SQL delete statement. After the process has completed, the table view refreshes displaying the reports that remain in the local database. If there are no reports in the local database, the user relocates to the home page.

"Purge Completed" removes the completed reports from the local database. An SQL select statement selects all reports that have been submitted and groups these primary numbers into in array. A for loop is used to delete these primary numbers in the array with an SQL delete statement.

4.6 Updating an Incomplete Report

An incomplete report is accessible to a reporter for data that have not been completed. The "Action" button has the same structure and functionality as the single report action button mentioned above.



Time/Place

Date

2011-09-27

Time

18:40:26

Geospatial

Latitude

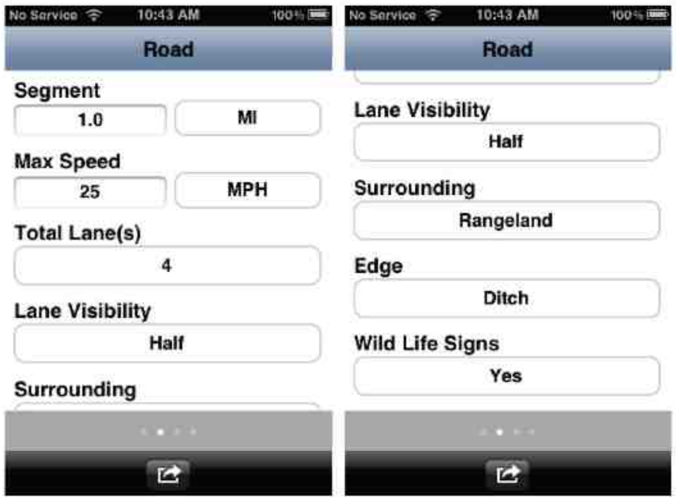
34.160925

Longitude

-119.044603

Figure 4.19 Retrieved Time/Place

The Time and Place category retrieves the user's date, time, latitude, and longitude of the report that was previously saved and inserted into their respective areas from the local database, as shown in Figure 4.19. These label values are not changeable.



Road

Segment: 1.0 MI

Max Speed: 25 MPH

Total Lane(s): 4

Lane Visibility: Half

Surrounding

Road

Lane Visibility: Half

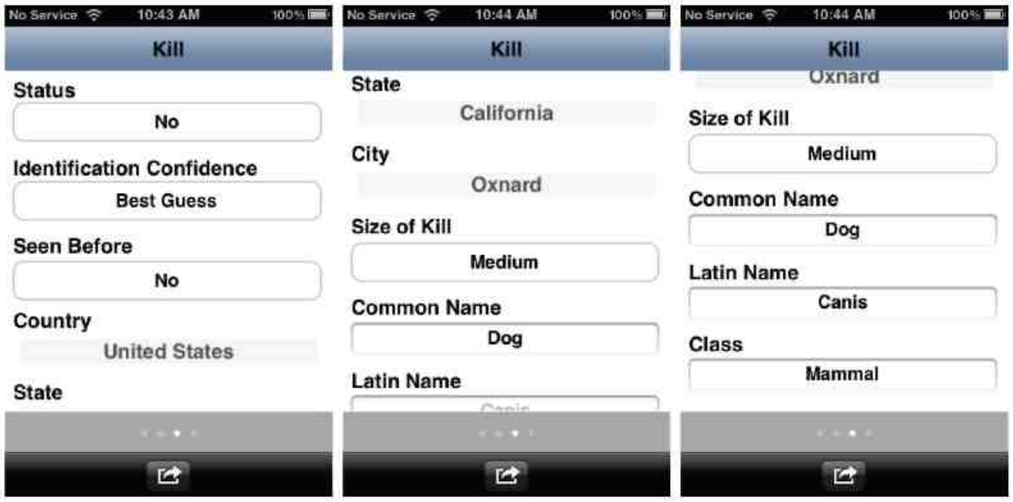
Surrounding: Rangeland

Edge: Ditch

Wild Life Signs: Yes

Figure 4.20 Updating Road Information

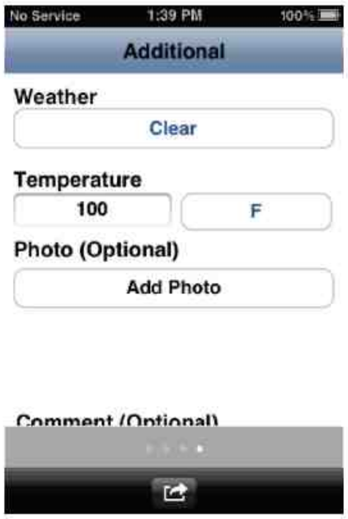
The Road category retrieves user information about the report that was previously saved and inserted into their respective areas from the local database, as shown in Figure 4.20. If a reporter has inserted values in the text fields, then those values are displayed. A reporter is permitted to change the information from the text fields. Changes made follow the same structure as the single report mentioned above for validating information for each text field. Picker values are retrieved from the database based on the reporter's selections. The values of these pickers are changeable.



Status
No
Identification Confidence
Best Guess
Seen Before
No
Country
United States
State
California
City
Oxnard
Size of Kill
Medium
Common Name
Dog
Latin Name
Canis
Class
Mammal

Figure 4.21 Updating Killed Information

The Kill category retrieves user information about the report that was previously saved and inserted into their respective areas from the local database, as shown in Figure 4.21. Country, state, and city are not changeable. If a reporter has inserted values in the text fields, then those values are displayed. Reporters are permitted to change the information from the text fields. Changes made follow the same structure as the single report mentioned above for validating information for each text field. Picker values are retrieved from the database based on the reporter's selections. The values of these pickers are changeable.

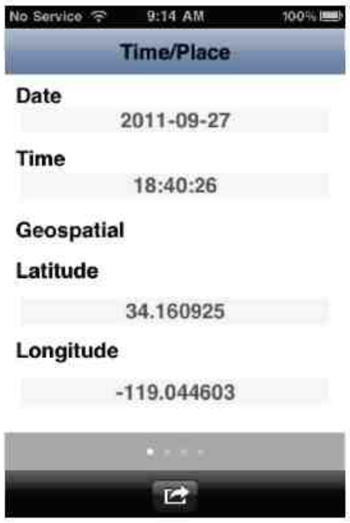


Weather
Clear
Temperature
100
F
Photo (Optional)
Comment (Optional)

Figure 4.22 Updating Supplementary Information

The Additional category retrieves user information about the report that was previously saved and inserted into their respective areas from the local database, as shown in Figure 4.22. Weather and temperature are not changeable. If an image was stored, then the image is retrieved from the local database and displayed. The image is changeable and has the same actions as the single report mentioned above. If a reporter has inserted a value in the text view, then that value is displayed. A reporter is permitted to change the information from the text view. Changes made follow the same structure as the single report mentioned above for validating information for the text view.

4.7 Retrieving a Completed Report



Date
2011-09-27
Time
18:40:26
Geospatial
Latitude
34.160925
Longitude
-119.044603

Figure 4.23 Retrieving a Completed Report

When the user selects a report from the table view of the home page, the primary number is transmitted to the complete view. The app executes an SQL select statement using the primary number received. After the app finds the report, it inserts the values into their respective areas from the local database, as shown in Figure 4.23. The app does not allow a user to modify any retrieved information in the report.

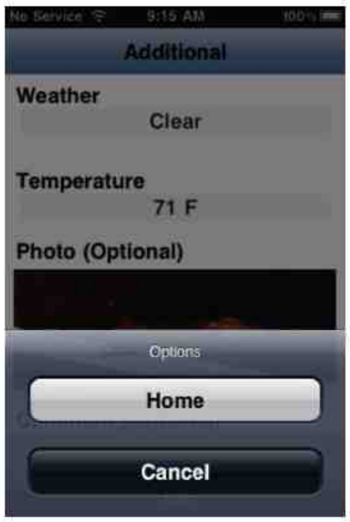


Figure 4.24 Actions for Completed Report

After the user has finished reviewing the completed report, the “Home” button relocates to the home page, as shown in Figure 4.24.

4.8 Badge Number Notification

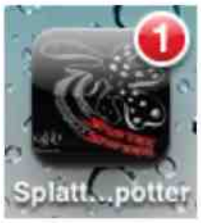
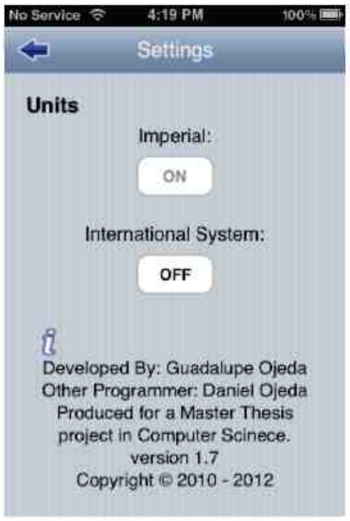


Figure 4.25 Badge Number Notification

The total number of incomplete reports in the local database determines the badge number. This number is shown on the app icons, as shown in Figure 4.25. The Badge function is called after reports are removed from the local database on the home page. When returning from any views to the main screen in the application, a badge function is executed updating the badge number. Primary numbers are grouped into an array through the use of an SQL select statement on the local database searching for incomplete reports. The total number of elements in the array is stored as the shared application number. This shared application number is displayed on the app icon as a badge.

4.9 Setting Default Units of Measurement


No Service 4:19 PM 100%

Settings

Units

Imperial: ON

International System: OFF

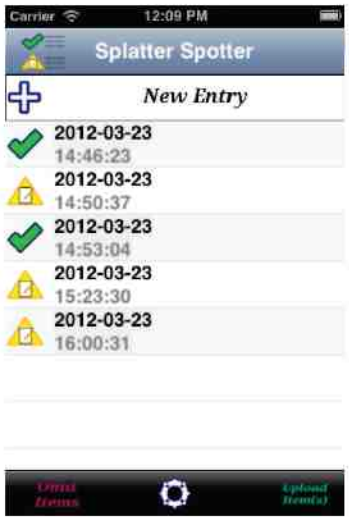
Developed By: Guadalupe Ojeda
Other Programmer: Daniel Ojeda
Produced for a Master Thesis
project in Computer Scinece.
version 1.7
Copyright © 2010 - 2012

Figure 4.26 Setting Default Units of Measurement

The "Settings" button is shown at the centered bottom of the home page view. Settings provide the user with a feature of adjusting the units of measurement in the app, as shown in Figure 4.26. There exist two types of units: Imperial and International System. Once the user has set their units of choice, these units are globally adjusted throughout the app with the use of user-shared variables.

Units are either "ON" or "OFF" in the settings page. On indicates that the units are enabled. Off indicates that the units are disabled. Both units cannot be enabled at the same time, as the app is either using Imperial or International system units. Fields that utilize these settings of units are segment, speed, and temperature. Segment has units MI (Miles) or KM (Kilometers). Speed has units MPH (Miles per hour) or KM/H (Kilometers per hour). Temperature has units F (Fahrenheit) or C (Celsius). These default settings are remembered even after the app is closed and removed from the multitasking section of the mobile device.

4.10 Functionalities of the Home Page



2012-03-23
14:46:23
2012-03-23
14:50:37
2012-03-23
14:53:04
2012-03-23
15:23:30
2012-03-23
16:00:31

Figure 4.27 Functionalities of the Home Page

The home page displays complete and incomplete reports, as shown in Figure 4.27. Users may also modify the reports that are stored locally by editing or deleting them. The New Entry button aids the reporter in initiating a report. Settings add an additional feature for setting global variables. The legend is positioned at the upper left of the home page view and depicts the icon for which each was intended.

The table view displays reports from the local database. The table view has a function that counts the number of rows in the database and presents that many entries. This table view retrieves data from the database and presents the date and time in one record cell. The date is displayed as the text label of the report while the time is displayed as the detail text label of the report. A report that has been submitted is accompanied with a green check mark. If the record has not been submitted, then a non-submitted image is shown instead. When a user selects a certain row from the table view, a primary number of that row is transmitted to the incomplete or complete view for further processing. Sending this primary number provides the correct record to be viewed.

Swiping left or right allows a user to delete a single row. In doing so, it displays a “Delete” button to the right of the row. After pressing the button, an SQL delete statement omits the report from the local database by using the primary number of that row.

Chapter 5: Conclusions

There were four goals that were accomplished in this project. These goals were to: create a functional mobile application, allow for transect observations, obtain crowd-source road kill information through an iPhone device, and assist in creating a repository of data related to road kill. Through the creation and implementation of the Splatter Spotter, this project was able to successfully accomplish all four goals. First, a functional mobile application was created to allow a reporter to input data into a single or a transect report. Reports could be stored to a local database on the device or to a remote database on the server. Reporters could view or modify existing reports that were stored on the mobile device. Second, transect observations were initiated to allow a reporter to observe a longer segment of the road. This type of observation utilizes the GPS feature built into the iPhone device to report various locations throughout the observed segment of road. Third, a crowd-sourcing system was accomplished in that road kill observations could be collected. Anyone with an iPhone device and the Splatter Spotter application could participate in collecting data about road kill. Fourth, a protocol was devised through the application that assisted in creating a repository of data related to road kill. All reports could be stored into one central location, allowing anyone to analyze the road kill data. This project could be transplanted to other crowd-sourcing mobile applications. For example, a botanist or entomologist can use what was done in this project to report significant information on plants/insects in the field. All it takes is an idea, and the willingness to produce an application.

Chapter 6: Future Work

A variety of additional modifications could be constructed if a developer were interested in improving this road kill application. A more aesthetic look for the application could be created for smoother user navigation. Alternative methods could be implemented for communication between client-side and server-side. Application features could be improved for better user interactions. An application could be created on alternative platforms and various tests could be performed for enhancements.

A newer design could allow the reporters to navigate through the application more efficiently. A "how to guide" could be created to assist reporters when they initially use the application. Retina could be integrated into the application, so that reporters can have a better viewing experience. An adjustment could be made to the code, so that the application can be used in either portrait or landscape mode. This would help in displaying images taken by the mobile device to be seen correctly in any orientation.

A secure communication protocol could be created to prevent snoopers from obtaining sensitive data when submitting to the server. Updating HTTP to HTTPS provides a more secure way of communicating between client-side and server-side. Exchanging information from server-side to client-side could also be improved by using JSON instead of XML. JSON could provide a faster, more reliable way for the application to parse the information from the server.

A feature could be created to permit a reporter to backup their local data on the mobile device. This could be accomplished by generating a website to download and upload backup files to and from the mobile device. Another backup solution could be the integration of iCloud that would allow reporters to backup local data to a remote location. Also, a function could be developed to allow reporters to sort/search reports by date, name, classification, complete, incomplete, and so on. An additional function could allow the reporters to visually display local data on a map using the iPhone device. Another function could allow the application to download a file from the server permitting the reporters to input road kill information faster. The file should contain animal common names and animal Latin names. This could be accomplished when reporters are inputting a common name or Latin name of the road kill to auto complete the remainder of the report.

The application could be built on another platform such as the iPad, Android, and Windows Phone to expand the reach of collecting data related to road kill. Such an application would allow an integration of various social networks for sharing and posting of road kill reports. Beta testing could be conducted to improve on any bugs that may exist. Also, stress testing could be conducted to simulate multiple uploads at the same time. The application could then be altered for a better user experience based on the results of the stress test.

References

1. Apache Software Foundation. (2012, June). *What is Apache Derby?*. Retrieved from http://db.apache.org/derby/#What+is+Apache+Derby.

2. Apple Inc. (n.d.). *Cocoa Touch*. Retrieved from http://developer.apple.com/technologies/ios/cocoa-touch.html.

3. Apple Inc. (2007, June). *iOS*. Retrieved from Apple website http://www.apple.com/ios/.

4. Apple Inc. (2012, June). *iOS 5.0*. Retrieved from http://developer.apple.com/library/ios/#releasenotes/General/WhatsNewIniPhoneOS/Articles/iOS5.html.

5. Apple Inc. (2012, June). *iOS Simulator*. Retrieved from http://developer.apple.com/library/ios/#documentation/Xcode/Conceptual/ios_development_workflow/25-Using_iOS_Simulator/ios_simulator_application.html#//apple_ref/doc/uid/TP40007959-CH9-SW1.

6. Apple Inc. (2009, June). *iPhone*. Retrieved from Apple website http://www.apple.com/iphone/.

7. Apple Inc. (2006, January). *Mac*. Retrieved from Apple website http://www.apple.com/mac/.

8. Apple Inc. (2007, October). *Objective-C*. Retrieved from http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html.

9. Apple Inc. (n.d.). *Register*. Retrieved from Apple website http://developer.apple.com/programs/register/.

10. Apple Inc. (2012, June). *Xcode*. Retrieved from Apple website http://developer.apple.com/xcode/.

11. CSUCI. (n.d.). *Splatter Spotter Overview*. Retrieved from CSUCI website http://roadkill.csuci.edu/.

12. erikreitan. (2008, June). *ASP.NET - Silverlight XAP FAQ*. Retrieved from http://forums.asp.net/t/1277554.aspx.

13. Google. (n.d.). *Download the Android SDK*. Retrieved from http://developer.android.com/sdk/index.html.

14. Google. (2011, November). *Galaxy Nexus 4G (LTE)*. Retrieved from http://www.android.com/devices/detail/galaxy-nexus-4g-lte.

15. Google. (n.d.). *Google Play Android Developer Console*. Retrieved from https://accounts.google.com/ServiceLogin?service=androiddeveloper&passive=true&nui=1&continue=https://play.google.com/apps/publish/&followup=https://play.google.com/apps/publish/.

16. Google. (2010, February). *Google weather api*. Retrieved from http://www.google.com/ig/api?weather=.

17. Google (2012, July). *XML layouts*. Retrieved from http://developer.android.com/guide/topics/ui/declaring-layout.html.

18. json. (1999, December). *Introducing json*. Retrieved from JSON website http://www.json.org/.

19. Kolowski, J., & Nielsen, C. (2008). Using penrose distance to identify potential risk of wildlife–vehicle collisions. *Biological Conservation*, 141, 1119-1128.

20. Lynda, Mapes V. (2012, March). *Animals on the move: first survey results, and photos galore*. Retrieved from http://seattletimes.nwsource.com/html/fieldnotes/2017670592_animals_on_the_move_first_survey_results_and_photos_galore.html.

21. Microsoft. (2010, November). *Discover*. Retrieved from http://www.microsoft.com/windowsphone/en-us/features/default.aspx.

22. Microsoft. (2012, March). *How to create your first silverlight application for windows phone*. Retrieved from http://msdn.microsoft.com/en-us/library/ff402526(v=vs.92).aspx.

23. Microsoft. (2012, March). *Windows Phone Development*. Retrieved from http://msdn.microsoft.com/en-us/library/ff402535(v=vs.92).aspx.

24. Microsoft Corporation. (n.d.). *Membership*. Retrieved from http://create.msdn.com/en-us/home/membership.

25. New Urban Mechanics. (2011, February). *Street Bump*. Retrieved from New Urban Mechanics website http://www.newurbanmechanics.org/projects/streetscapes/bump/.

26. Ojeda, D. (2012). *Web-based reporting system for road kill*.

27. Osterhues, Marlys. (2008, October). *Best practices manual: wildlife vehicle collision reduction study*. Retrieved from http://environment.fhwa.dot.gov/ecosystems/wvc/ch2.asp.

28. SQLite. (2000, August). *SQLite*. Retrieved from SQLite website http://www.sqlite.org/.

29. SQLite. (n.d.). *Sqlite versus dbstar*. Retrieved from http://www.sqlite.org/cvstrac/wiki?p=SqliteVersusDbstar.

30. SQLite. (n.d.). *Sqlite versus derby*. Retrieved from http://www.sqlite.org/cvstrac/wiki?p=SqliteVersusDerby.

31. SQLite. (n.d.). *Sqlite versus firebird*. Retrieved from http://www.sqlite.org/cvstrac/wiki?p=SqliteVersusFirebird.

32. W3C. (2012, January). *XML technology*. Retrieved from W3C website http://www.w3.org/XML/.

33. W3schools. (n.d.). *JSON*. Retrieved from http://www.w3schools.com/json/default.asp.

34. Wikipedia. (2012, July). *Android software development*. Retrieved from http://en.wikipedia.org/wiki/Android_software_development.

35. Wikipedia. (2012, July). *APK (file format)*. http://en.wikipedia.org/wiki/APK_(file_format).

36. Wikipedia. (2012, July). *Application software*. Retrieved from http://en.wikipedia.org/wiki/Application_software.

37. Wikipedia. (2012, July). *C sharp (programming language)*. Retrieved from http://en.wikipedia.org/wiki/C_Sharp_(programming_language).

38. Wikipedia. (2012, July). *Cocoa touch*. Retrieved from http://en.wikipedia.org/wiki/Cocoa_Touch.

39. Wikipedia. (2012, July). *Communications protocol*. Retrieved from http://en.wikipedia.org/wiki/Communications_protocol.

40. Wikipedia. (2012, July). *Crowdsourcing*. http://en.wikipedia.org/wiki/Crowdsourcing.

41. Wikipedia. (2012, July). *Database*. Retrieved from http://en.wikipedia.org/wiki/Database.

42. Wikipedia. (2012, July). *Development*. Retrieved from http://en.wikipedia.org/wiki/Windows_Phone#Development_2.

43. Wikipedia. (2012, July). *Extensible application markup language*. Retrieved from http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language.

44. Wikipedia. (2012, July). *HTTP secure*. Retrieved from http://en.wikipedia.org/wiki/HTTP_Secure.

45. Wikipedia. (2012, July). *iPhone*. Retrieved from http://en.wikipedia.org/wiki/IPhone.

46. Wikipedia. (2012, July). *iPhone 3GS*. Retrieved from http://en.wikipedia.org/wiki/IPhone_3GS.

47. Wikipedia. (2012, July). *iOS*. Retrieved from http://en.wikipedia.org/wiki/IOS_(Apple).

48. Wikipedia. (2012, July). *Java (programming language)*. Retrieved from http://en.wikipedia.org/wiki/Java_(programming_language).

49. Wikipedia. (2012, July). *Mac os x*. Retrieved from http://en.wikipedia.org/wiki/Mac_OS_X.

50. Wikipedia. (2012, June). *Mobile device*. Retrieved from http://en.wikipedia.org/wiki/Mobile_device.

51. Wikipedia. (2012, July). *SQLite*. Retrieved from http://en.wikipedia.org/wiki/SQLite.

52. Wikipedia. (2012, July). *System software*. Retrieved from http://en.wikipedia.org/wiki/System_software.

53. Wikipedia. (2012, July). *Windows nt*. Retrieved from http://en.wikipedia.org/wiki/Windows_NT.

54. Wikipedia. (2012, July). *Xcode*. Retrieved from http://en.wikipedia.org/wiki/Xcode.

55. Wikipedia. (2012, July). *XML*. Retrieved from http://en.wikipedia.org/wiki/Xml.

56. Wikipedia. (2012, July). *XNA build*. Retrieved from http://en.wikipedia.org/wiki/Microsoft_XNA#XNA_Build.

Appendix A: Database Entity Set Description

This appendix discusses the database table and the attributes that hold the values for the users.

A.1 Entity: roadkill

1. Description: The Roadkill table holds information about the survey taken by the user in the Splatter Spotter app.

2. Attributes:

    1. Name: rkid
    2. Description: Holds the identification number of each survey.
    3. Domain/Type: integer
    4. Value-Range: 0 – 9999999999
    5. Signed, Unsigned, None: None
    6. Default value: None
    7. Null Value Allowed or Not: Not
    8. Unique: Yes
    9. Single or Multiple-Value: Single
    10. Simple or Composite: Simple

    1. Name: currdate
    2. Description: Specify the date of observation.
    3. Domain/Type: varchar
    4. Value-Range: 10
    5. Signed, Unsigned, None: None
    6. Default value: '0000–00–00'
    7. Null Value Allowed or Not: Not
    8. Unique: No
    9. Single or Multiple-Value: Single
    10. Simple or Composite: Simple

    1. Name: currtime
    2. Description: Specify the time of observation.
    3. Domain/Type: varchar
    4. Value-Range: 8

5. Signed, Unsigned, None: None
6. Default value: ‘00:00:00’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: latitude
2. Description: Specify latitude location of observation.
3. Domain/Type: varchar
4. Value-Range: 8 – 10
5. Signed, Unsigned, None: None
6. Default value: ‘90.000000’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: longitude
2. Description: Specify longitude location of observation.
3. Domain/Type: varchar
4. Value-Range: 8 – 11
5. Signed, Unsigned, None: None
6. Default value: ‘180.000000’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: segmentnum
2. Description: Specify the observed distance of the segment.
3. Domain/Type: varchar
4. Value-Range: 3 – 5
5. Signed, Unsigned, None: None
6. Default value: ‘0.0’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: segmenttype
2. Description: Specify the unit type of the segment observed.
3. Domain/Type: varchar
4. Value-Range: 2
5. Signed, Unsigned, None: None
6. Default value: 'MI'
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: MaxSpeednum
2. Description: Specify the cars max speed at observation.
3. Domain/Type: varchar
4. Value-Range: 3 – 5
5. Signed, Unsigned, None: None
6. Default value: '0.0'
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: MaxSpeedtype
2. Description: Specify the unit type of max speed.
3. Domain/Type: varchar
4. Value-Range: 3 – 4
5. Signed, Unsigned, None: None
6. Default value: 'MPH'
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Lanes
2. Description: Specify how many lanes existed.
3. Domain/Type: varchar
4. Value-Range: 1 – 2
5. Signed, Unsigned, None: Unsigned
6. Default value: '1'
7. Null Value Allowed or Not: Not

8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Lane_Visibility
2. Description: Specify if all or half of the lanes were visible.
3. Domain/Type: varchar
4. Value-Range: 3 – 4
5. Signed, Unsigned, None: None
6. Default value: ‘All’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Surrounding
2. Description: Specify the surrounding.
3. Domain/Type: varchar
4. Value-Range: 5 – 9
5. Signed, Unsigned, None: None
6. Default value: ‘Urban’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Edge
2. Description: Specify the edge.
3. Domain/Type: varchar
4. Value-Range: 4 – 12
5. Signed, Unsigned, None: None
6. Default value: ‘Cliff’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Wildlife_Signs
2. Description: Specify if there were wildlife signs.
3. Domain/Type: varchar

4. Value-Range: 2 – 3
5. Signed, Unsigned, None: None
6. Default value: ‘Yes’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Kill_Status
2. Description: Specify if the animal is dead or alive.
3. Domain/Type: varchar
4. Value-Range: 2 – 3
5. Signed, Unsigned, None: None
6. Default value: ‘Yes’
7. Null Value Allowed or Not: Null
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: ID_Confidence
2. Description: Specify the confidence level of the observation.
3. Domain/Type: varchar
4. Value-Range: 3 – 10
5. Signed, Unsigned, None: None
6. Default value: ‘Definite’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Seen_Kill_Before
2. Description: Specify if the animal has been observed before.
3. Domain/Type: varchar
4. Value-Range: 2 – 3
5. Signed, Unsigned, None: None
6. Default value: ‘Yes’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Country
2. Description: Specify the country where the animal was observed.
3. Domain/Type: varchar
4. Value-Range: 1 – 255
5. Signed, Unsigned, None: None
6. Default value: ‘’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: State
2. Description: Specify the state where the animal was observed.
3. Domain/Type: varchar
4. Value-Range: 1 – 255
5. Signed, Unsigned, None: None
6. Default value: ‘’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: City
2. Description: Specify the city where the animal was observed.
3. Domain/Type: varchar
4. Value-Range: 1 – 255
5. Signed, Unsigned, None: None
6. Default value: ‘’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple
1. Name: Kill_Size
2. Description: Specify the animal size at observation.
3. Domain/Type: varchar
4. Value-Range: 5 – 6
5. Signed, Unsigned, None: None
6. Default value: ‘Small’
7. Null Value Allowed or Not: Not
8. Unique: No

9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Kill_Species_Common_Name
2. Description: Specify the animal's common name.
3. Domain/Type: varchar
4. Value-Range: 1 – 255
5. Signed, Unsigned, None: None
6. Default value: ‘’
7. Null Value Allowed or Not: Allowed
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Kill_Species_Latin_Name
2. Description: Specify the animal's Latin name.
3. Domain/Type: varchar
4. Value-Range: 1 – 255
5. Signed, Unsigned, None: None
6. Default value: ‘’
7. Null Value Allowed or Not: Allowed
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Kill_Class
2. Description: Specify the animal's classification.
3. Domain/Type: varchar
4. Value-Range: 1 – 255
5. Signed, Unsigned, None: None
6. Default value: ‘’
7. Null Value Allowed or Not: Allowed
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Weather
2. Description: Specify the weather at observation.
3. Domain/Type: varchar
4. Value-Range: 3 – 23

5. Signed, Unsigned, None: None
6. Default value: 'Clear'
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Temperature
2. Description: Specify the temperature at observation.
3. Domain/Type: varchar
4. Value-Range: 3 – 6
5. Signed, Unsigned, None: None
6. Default value: '70 F'
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Photo
2. Description: Optional photo if user wants.
3. Domain/Type: blob
4. Value-Range: 0 – size of data inserted
5. Signed, Unsigned, None: Unsigned
6. Default value: 'nil'
7. Null Value Allowed or Not: Allowed
8. Unique: Yes
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Comments
2. Description: Optional comments if user wants.
3. Domain/Type: varchar
4. Value-Range: 1 – 255
5. Signed, Unsigned, None: None
6. Default value: ''
7. Null Value Allowed or Not: Allowed
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: All_Fin
2. Description: Check if the user has input all fields.
3. Domain/Type: tinyint
4. Value-Range: 0 – 1
5. Signed, Unsigned, None: None
6. Default value: ‘0’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Submitted
2. Description: Check if the user has submitted the entry.
3. Domain/Type: tinyint
4. Value-Range: 0 – 1
5. Signed, Unsigned, None: None
6. Default value: ‘0’
7. Null Value Allowed or Not: Not
8. Unique: No
9. Single or Multiple-Value: Single
10. Simple or Composite: Simple

1. Name: Transect_Points
2. Description: Keeps origin, way, and destination points of the road being observed for transect.
3. Domain/Type: text
4. Value-Range: 0 – 65,535
5. Signed, Unsigned, None: None
6. Default value: ‘’
7. Null Value Allowed or Not: Allowed
8. Unique: No
9. Single or Multiple-Value: Multiple
10. Simple or Composite: Composite

3. Candidate Key: rkid
4. Primary Key: rkid
5. Strong/Weak Entity: Strong
6. Field to be indexed: None