DESIGN AND IMPLEMENTATION

OF

CREEL SURVEY DATA MANAGEMENT APPLICATION

A Project

Presented to

The Faculty of the Graduate School

At the University of Missouri

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

Implemented and Defended by:

Nishant Chettri

Prof. Yi Shang, Advisor

December 2017

TABLE OF CONTENTS

[List of Figures iii](#_Toc498184339)

[List of Tables vi](#_Toc498184340)

[Acknowledgements vii](#_Toc498184341)

[Abstract viii](#_Toc498184342)

[1. Introduction 1](#_Toc498184343)

[1.1. Data Collection Applications 1](#_Toc498184344)

[1.1.1. MDC Shooting Range Application 3](#_Toc498184345)

[1.1.2. MDC Bow Hunting Observation Application 3](#_Toc498184346)

[1.2. Creel Survey 3](#_Toc498184347)

[1.3. Problem Description 4](#_Toc498184348)

[1.4. Proposed Solution for Creel Survey 6](#_Toc498184349)

[2. Background and Related Works 9](#_Toc498184350)

[3. Design 13](#_Toc498184351)

[3.1. Design Theory 13](#_Toc498184352)

[3.2. System Architecture 14](#_Toc498184353)

[3.2.1. Creel Survey Mobile Application Architecture 14](#_Toc498184354)

[3.2.2. Creel Survey Dashboard Architecture 19](#_Toc498184355)

[3.3. Data Design 21](#_Toc498184356)

[3.3.1. Creel Survey Mobile Application Data Design 21](#_Toc498184357)

[3.3.2. Creel Survey Dashboard Data Design 23](#_Toc498184358)

[3.4. Technology Stack 24](#_Toc498184359)

[3.4.1. Front-End and Client Side 25](#_Toc498184360)

[3.4.2. Back-End and Server Side 25](#_Toc498184361)

[3.4.3. Other utilities 25](#_Toc498184362)

[4. Implementation 27](#_Toc498184363)

[4.1. Creel Survey Mobile Application 27](#_Toc498184364)

[4.1.1. Survey Module 28](#_Toc498184365)

[4.1.2. Data Collection Module 31](#_Toc498184366)

[4.1.3. Networking Module 34](#_Toc498184367)

[4.1.4. Notable Features on Creel Survey Mobile Application 38](#_Toc498184368)

[4.2. Creel Survey Web Dashboard 41](#_Toc498184369)

[4.2.1. Web Application File Hierarchy 42](#_Toc498184370)

[4.2.2. Account Management Module 43](#_Toc498184371)

[4.2.3. Data Verification Module 45](#_Toc498184372)

[4.2.4. Project Management Module 51](#_Toc498184373)

[4.2.5. User Management Module 53](#_Toc498184374)

[4.2.6. Utilities 55](#_Toc498184375)

[4.2.7. Notable Features on Creel Survey Web Dashboard 61](#_Toc498184376)

[5. Results 63](#_Toc498184377)

[6. Conclusion and Future Work 65](#_Toc498184378)

[6.1. Contribution and Achievement 65](#_Toc498184379)

[6.2. Future Work 66](#_Toc498184380)

[7. References 67](#_Toc498184381)

# List of Figures

Figure 1: Data Collection Application Workflow [4] 2

Figure 2: MDC Shooting Range Mobile and Web Application 3

Figure 3: MDC Creel Survey Data Sheet 5

Figure 4: Iterative Model [10] 6

Figure 5: MDC Shooting Range Mobile and Web Dashboard Homepages 10

Figure 6: MDC Bow Hunting Mobile Application Interfaces 11

Figure 7: Phonegap APIs [27] 16

Figure 8: G2 Crowd Grid® for Mobile Development Frameworks [25] 16

Figure 9: Cordova Architecture [12] 17

Figure 10: Creel Survey Dashboard Architecture Diagram 20

Figure 11: Creel Survey Mobile Application Data Design Tree 23

Figure 12: Creel Survey Dashboard Data Design Tree 24

Figure 13: Creel Survey Mobile Application Implementation 27

Figure 14: Creel Survey Mobile Application. Login; Homepage; Angler View; Tabular View 28

Figure 15: Party size initialization 29

Figure 16: Cordova camera plugin code 30

Figure 17: Fish caught view with fish image as thumbnail 30

Figure 18: Single vs Nested JSON array stored in HTML5 local storage 32

Figure 19: Dynamically insert data into SQLite database 33

Figure 20: Cordova plugin to check network connection; Download data from server to device 35

Figure 21: Upload data to server 36

Figure 22: PHP Parsing uploaded data and executing insert queries 37

Figure 23: Copy-to-all function 38

Figure 24: Spinner when loading 39

Figure 25: Practice and Previous survey view 40

Figure 26: Creel Survey Mobile Application folder structure 40

Figure 27: Creel Survey Web Dashboard Implementation 41

Figure 28: Web Application File Hierarchy 42

Figure 29: User and Roles table schema 43

Figure 30: Login and Registration Interface 44

Figure 31: Salting and Hashing a password 45

Figure 32: Administrator Permission Level validation 45

Figure 33: Unauthorized access redirection 46

Figure 34: Web Dashboard Homepage 46

Figure 35: Web Dashboard Treeview and Information Panel] 47

Figure 36: Party Verification Module 49

Figure 37: Angler Verification Module 50

Figure 38: Project List View 51

Figure 39: PHP Code to insert data to the project-list table 52

Figure 40: Add Project View 53

Figure 41: Edit Project View 53

Figure 42: User List View 54

Figure 43: Add User View 54

Figure 44: Edit User View 54

Figure 45: Project-User Associative ERD 55

Figure 46: Upload CSV Interface 56

Figure 47: Translating first line of CSV to JSON array 57

Figure 48: Insert CSV into database using LOAD DATA INFILE syntax 57

Figure 49: Download CSV Interface 58

Figure 50: Request for schedule handler and action-handling 59

Figure 51: PDO SQL Transactions 60

Figure 52: Random Number Generator 61

Figure 53: View Raw Angler tables - Datatable implementation 62

# List of Tables

Table 1: Native vs. Hybrid Mobile Application [24] 15

Table 2: Difference between paper-based logging and CSDMA 64

# Acknowledgements

Firstly, I would like to thank Prof. Yi Shang for his mentorship, guidance and support throughout my graduate school and through this research project. I would also like to thank Tom, Martha, Sherry and Joel for the opportunity to collaborate with the Missouri Department of Conservation and their feedback on this project.

I would also like to thank everyone from the Computer Science lab, especially Nickolas Wergeles, Zeshan Peng and William Morrison, for sharing their experiences and being an integral part of this project. I would also like to show appreciation to Jodie Lenser and Shirley Holdmeier for guiding me through my application process to Mizzou as well as advising me on my courses.

I would also like to thank Dr. Keshav Bhattarai, who inspired my application to Mizzou. Without his helpful advice, guidance and support throughout my application process, I would not have been a part of Mizzou.

Finally, I would like to thank my family. Without their support, trust and belief in me, I would not be where I am today. I hope I have done you proud.

# Abstract

Over the last 10 years, smartphones and other mobile devices have become extremely important, not only in everyday use, but also in science, politics, sports, education, and so many other fields. Harnessing the power of these mobile devices–the portability, convenience, and real-time access to data, mobile data collection and management have been developed in many applications to collect and manage tremendous amounts of real-life data in structured, interactive, user-friendly, and oftentimes, real-time fashion. In this project, an integrated mobile data collection and web-based data management system, called *Creel Survey Data Management Application (CSDMA)*, has been successfully designed and developed for the Missouri Department of Conservation. *CSDMA* is comprised of two major components: 1) the *Creel Survey Mobile Application* – a Hybrid application developed using Phonegap and Apache Cordova, which collect survey data on an iPad or an iPhone, and uploads them to a MySQL database on an Amazon Web Services (AWS) EC2 instance, and 2) the *Creel Survey Web Dashboard* developed using LAMP stack, which organizes and displays the aforementioned data for biologists and conservation staffs to view and verify in real-time using a user-friendly web interface developed using the Bootstrap CSS3 framework. *CSDMA* provides a convenient and efficient tool for conservation staffs to collect substantial amounts of angler (fishermen and fisherwomen) survey data over an extended amount of time, e.g., months at a time, at many parks and conservation locations throughout Missouri. The system stores the data in an organized manner and supports web-based interaction with the data in real-time. CSDMA successfully transforms the current paper-based, tedious and error-prone surveying practice to a digital, automated and real-time process, enabling conservation staffs to carry out surveys much more efficiently and analyze data much more easily and in real-time.

# Introduction

Over the past decade, mobile devices and smart phones have become extremely pervasive. According to the study conducted by the Pew Research Center in January 2017, 77% of the people now own a smart phone in the United States [1]. Ubiquitous access to smart devices has enabled field-based studies to utilize them for data collection and analysis. Researchers can now use these devices in the field to collect data and send it to a server where it can be analysed in real-time expediting the entire process. Also, with most of the users having their own smart devices, this makes it more convenient [2] reducing the overall cost of the process, and without any personnel having to re-type the collected information, data can be more accurate, invariably reducing the overall cost in terms of time and human resources [3].

## Data Collection Applications

Science and research is driven by data. As more data is collected, scientists are able to learn more about a particular subject. This applies to areas such as healthcare, sports, politics, and so on. And with the hardware and software improvements in computational devices such as mobile phones and tablets, collection of data has been aided in ways previously unimagined.

Mobile phones have evolved from simple machines used for making phone calls and sending text messages to powerful devices capable of providing desktop-level computational power. Improvement in mobile devices have enabled users to take high quality pictures, send data over networks, visualize content on-the-go, and so much more. And combining some of the vast number of functionalities, the amalgamation of technology and science has led to a more advanced research environment.

In the field of data collection, mobile devices have allowed researchers to go into the field, capture, store, update, and analyze data faster than ever before. They have become a global standard due their presence in everyday life and the ability of mobile applications to simplify the work being done by researchers. The data being collected in the field can be sent to various parts of the world in real-time for a more collaborative engagement. Data and research can be conducted in a real-world scenario, which provides a more accurate and informed data collection process.

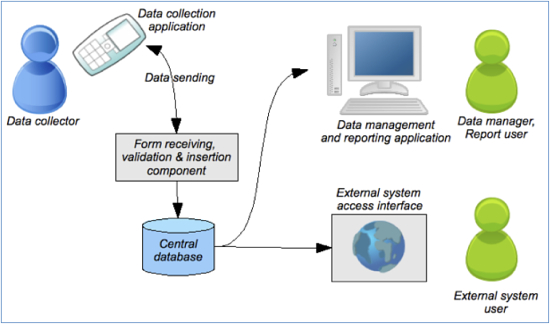


Figure 1: Data Collection Application Workflow [4]

Data uploaded by on-site researchers and data collectors can consequently be visualized at a different location by analysts through a web based application. Rather than using paper-based reports and reporting tools on desktops such as MS-Excel, web applications provide better multi-user interaction, more real-time functionalities, portability and convenience to the analysts. Web applications abstracts the non-essential information to the users while allowing them to visualize the critical data, verify its validity, and apply statistical operations on those data to obtain further information.

### MDC Shooting Range Application

MDC Shooting Range Application is a data collection application used by Missouri Department of Conservation (MDC) to gather information about shooting ranges within Missouri. This application has two major components: 1) a mobile application, where data collection was done in the field via Android device and sent to a remote server database, and 2) a web application, where data sent to the database are displayed for further analysis [5].

Figure 2: MDC Shooting Range Mobile and Web Application

### MDC Bow Hunting Observation Application

Another collaboration with MDC led to the development of a mobile application to gather information about bow hunting activities in Missouri. A prototype was developed using Hybrid mobile development technology for Android/iOS devices, which has been used as a driving factor for CSDMA.

## Creel Survey

MDC Fisheries Department conduct Creel Surveys to collect information about fishing activities in Missouri. Information such as number of anglers (fishermen and fisherwomen), fishing success, catch and harvest rates, angler preferences, and so on help them create a better fishing environment. To obtain such information surveys are carried out. These surveys are conducted with the help of on-site *clerks* taking interviews of anglers, *biologists* analyzing the data obtained by the clerks and verifying the validity of the data, and *MDC staff*, who are responsible for managing the administrative aspect of the survey, e.g., creating questionnaires, assigning clerks to their respective projects, and so on.

Clerks conduct surveys at various sites whenever a survey is scheduled. They interview angler parties (group of anglers) and collect information about each of their fishing habits, the details of the fish caught, and other survey details. The purpose of Creel Survey is to regulate the fishing activity, monitor the fishing trends, evaluate the fishery management (e.g., improvement of fishing habitats) and asses the success of fishing in Missouri [6]. There are 3 types of surveys conducted in the Missouri: Access, Stream aka Bus Route, and Roving Creels. Interviews are conducted at the various accesses (exit points) of a lake. The surveys can be separated based on the number of accesses and the usage of each access point. MDC clerks wait for and conduct their interview at these access points. The responses are recorded on paper and taken back to MDC, where they are manually entered into a database.

## Problem Description

As useful as the process of data collection is, it is arduous and time-consuming, especially when they are done on paper. Current survey process involves clerks obtaining multiple sheets of paper (data sheets) from MDC and taking interviews at the survey sites. Each angler party interview represents a single data-sheet, as shown in Figure 3.

The clerks conduct interviews every day, and depending on the number of anglers at the sites, this produces copious amounts of data. MDC staff examine all those surveys once again, validate it and pass it onto data entry personnel, who enter the data into the databases manually. Depending on the amount of data, it could take anywhere between 15-20 hours per month to enter them into the database, making it a slow, redundant and arduous process, which invariably wastes time, human resource hours, and money. Moreover, this process can be prone to mistakes and errors since the data entry personnel may not be aware of specific survey details, which may have been missed during the validation process, and could be entered incorrectly into the database. Moreover, data sheets can be misplaced, lost or damaged by the time it goes from the clerk to the data entry personnel. As such, the entire workflow is flawed.

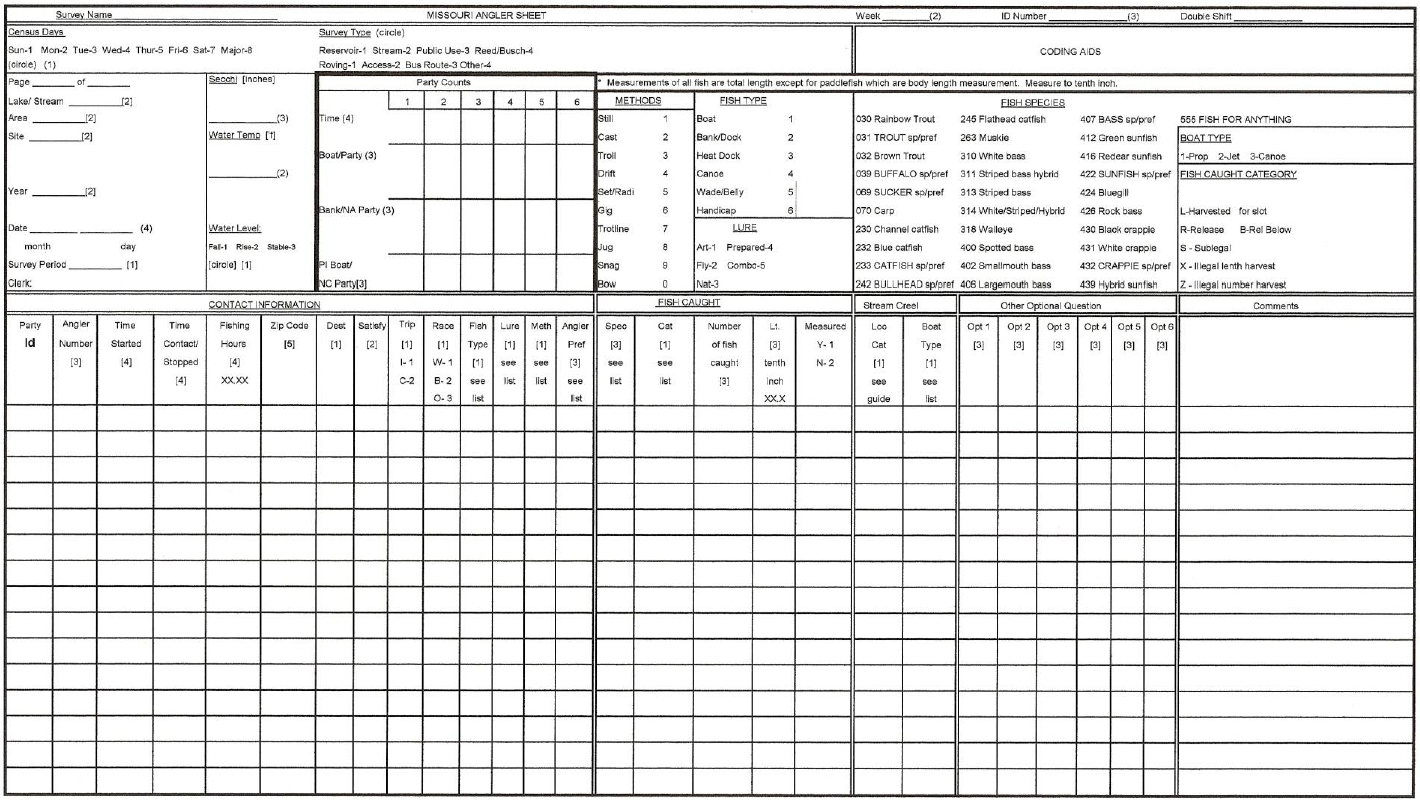


Figure 3: MDC Creel Survey Data Sheet

Real-time data provides a quicker insight into the collected information, and enables researchers to identify innovative ways to understand the data earlier than before [7]. It can help identify any flaws in the workflow during the process and apply fixes before any or further data loss takes place [8]. As the current process of data collection in MDC does not occur in real-time, there is a gap between each of the data collection, verification and analysis stage. Furthermore, this creates delays in current and future surveys, making it difficult to verify or correct the workflow. If there is to be an addition or modification at any stage of the process, it will likely be done after the entire survey cycle is completed, and does not accommodate making changes when the process is underway in terms of changing questions, modifying database schema, addition of new sites or fields in the survey as it could possibly create an issue with the previously conducted surveys before these changes are implemented.

## Proposed Solution for Creel Survey

To address these various problems a data collection mobile application, and a data verification web dashboard were proposed in this project. The mobile application is used in the field to collect data and upload it directly to the database, while the dashboard is used for analyzing and verifying the data in real-time. These two modules were implemented using a software engineering approach using the Iterative model [9].

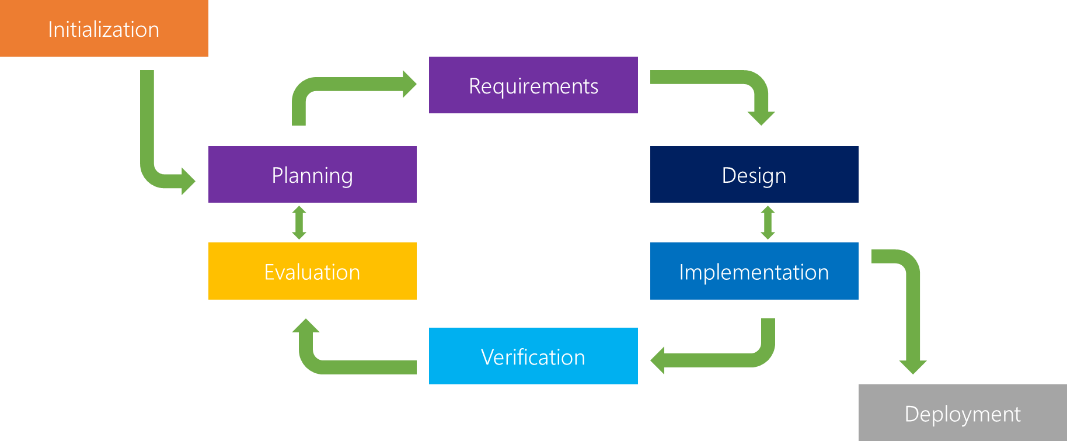


Figure 4: Iterative Model [10]

The project initially had a clear overview of its essential goals but was still unclear in the detailed functionality requirements. Moreover, the project was being developed using a novel hybrid mobile development. To accommodate the changing requirements, and need for quicker implementation and deployment, the Iterative methodology was the most suitable option [11]. Also, since the client was able to meet on a regular basis, frequent client interaction to provide updates, get feedback and produce iterations was not an issue. Weekly and on occasion bi-weekly meetings were held with the MDC staff to gather information about their requirements, update them on the design and development stage of the application, improve on any usability feedback, and append any additional functionality requirements identified by them.

As mentioned previously, there are two major components of this project: a mobile data collection application to be developed for an iPad, and a web application for data analysis and verification.

The mobile application was developed using Apache Cordova with Adobe PhoneGap, which is a hybrid mobile application development framework [12]. Initially, it was a possibility where the application would be distributed to both iOS and Android devices, which made using the Hybrid mobile development approach the most appropriate choice, as development with it allows distribution of an application on both those platforms simultaneously. Utilizing the Phonegap framework for Hybrid mobile development along with Cordova, involves creating a mobile-based web application, which is launched as a native application. The mobile application was developed using HTML5, Bootstrap with CSS3, and Javascript with jQuery libraries, and SQLite was used for persistent storage on the device. Once data is collected on the mobile application, data is uploaded to a remote server hosting a centralized database, which is MySQL, an open-sourced relational database management system.

The web dashboard, hosted on the same server as the database, was developed using the LAMP stack (Linux, Apache web server, MySQL database, and PHP). The website provides MDC the ability to verify data in real-time with the help of the database, as well as functionalities such as adding new projects, uploading and download data through CSV files. To protect the sensitive data, security measures such as Salted-Hashed passwords [13], user roles and access restrictions based on those user roles were also applied on the web dashboard.

The mobile and web-based solution improves the slow and laborious process of collecting survey data through a bulky paper-based system to a rapid, user-friendly and convenient data collection process with the ability to view, verify and analyze real-time data. Any errors or flaws identified can be easily accommodated through the database by adding or removing fields in the tables. Moreover, with clerks being able to upload data directly from the mobile application to the server, there is no need for any data entry personnel either saving time and money, as well as allowing MDC to redirect their human resource for other tasks.

The remainder of the report is organized as follows: In Chapter 2, related works in data collection and their strengths and limitations are discussed. The design theory of the mobile application and web dashboard are discussed in Chapter 3. Chapter 4 presents the implementation of the mobile application and web dashboard. The results of implementing these applications are discussed in Chapter 5. The report is concluded in Chapter 6 and discusses potential future works

# Background and Related Works

Data collection applications enable a more convenient method of conducting surveys and interviews. They are also capable of adding dimensions to the applications previously unexplored. Integrating a web dashboard with the mobile data collection adds further possibilities. As such, similar systems need to be researched to identify useful tools and techniques, which can be used as part of the solution. This chapter discusses some mobile data collection applications, their relevance to the proposed solution.

[14] discusses the significance of geoscientists using mobile applications over data sheets to collect data. It presents how transcribing data from collected data sheets can be error-prone or lost due to misplaced or damaged data sheets. The paper introduces *High Density Large Wood Debris* (*HDLWD) Effectiveness* application to illustrate the effectiveness and the scalability provided by using mobile apps as compared to data sheets as well as added functionalities of taking videos and images, which are not possible when using data sheets. CSDMA integrates some of these features and enables MDC to collect data at a larger scale than ever before.

Moreover, applications such as [15], have provided features for anglers themselves to report their fishing activities, along with details about the fish that are caught during those periods. However, as MDC requires their own staff to conduct these interviews, mapping it similar to this application would be counterproductive. However, image capture feature provided in this application was an inspiration for the proposed solution. Previous research into popular fishing applications have revealed the efficiency and cost-effective nature of mobile applications in exploring long-term and broad trends in angler demographics [16]. Analysis of 3 years of angler data collected from popular mobile fishing applications have led to exploration of disease outbreaks, fish stocking events, responses to changing regulations and so on. The purpose of conducting creel surveys is to gather data to monitor similar details about the lakes in Missouri.

*MDC Shooting Range* was an application developed by University of Missouri – Columbia for MDC’s Natural Resource Management and is one of the major inspirations for the proposed application. This application has two major components: 1) a mobile application, where data collection was done in the field via Android device and sent to a remote server database, and 2) a web application, where data sent to the database are displayed for further analysis. The purpose of the application was to enable clerks and surveyors to examine the quality and status of shooting ranges in Missouri [5].

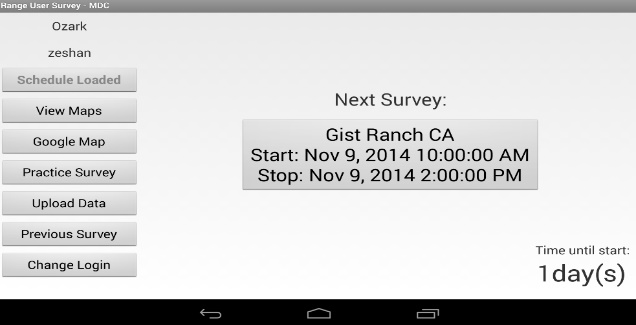


Figure 5: MDC Shooting Range Mobile and Web Dashboard Homepages

The application’s *Data Collection* module stores all survey data in SQLite database, which provides a convenient format of storing data, similar to a relational database, which allows organized storage and retrieval of data on any device. Similarly, the database on the servers are also using a Relational Database Management System, MySQL. All the technology used here are available free and open-source, which makes it a cost-effective solution. As MDC found the application extremely convenient and useful and since the proposed solution was in essence similar in implementation, functionalities such as *Treeviews* were borrowed and improved upon through *Lazy Loading*. The server was hosted on Amazon’s Elastic Compute Cloud (EC2), as a Linux based instance, which is a popular and convenient method of hosting data and web dashboard on the cloud.

Similarly, *Bow Hunting Application* was a prototype created to conduct on-site surveys of Bowhunters throughout the state of Missouri, and under the supervision of MDC. It involved a similar data collection process but was developed using the cross-platform (Hybrid) mobile development approach through Apache Cordova. Hybrid applications are developed using web development technologies and can be deployed to multiple mobile OS platforms (iOS, Android, Windows, etc.) with minimal change to the code. This allows for rapid multi-platform deployment. The application used HTML5’s Web Storage to store survey data. The application was less computationally and data intensive, and the capability to store data in the form of Web Storage’s *Local storage* objects was used, which allows it to store information reminiscent of cookies in web browsers [17]. This is useful when passing information between views in the application without having to access a database, making it quicker and easier to retrieve data on the client-side.

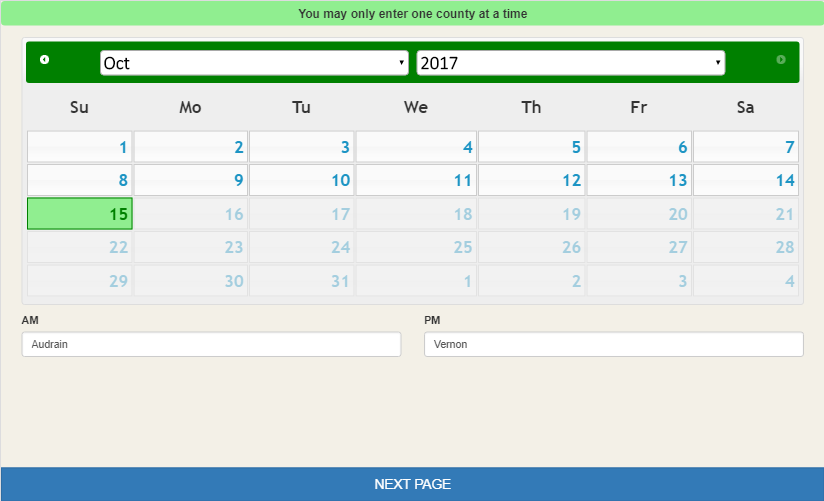
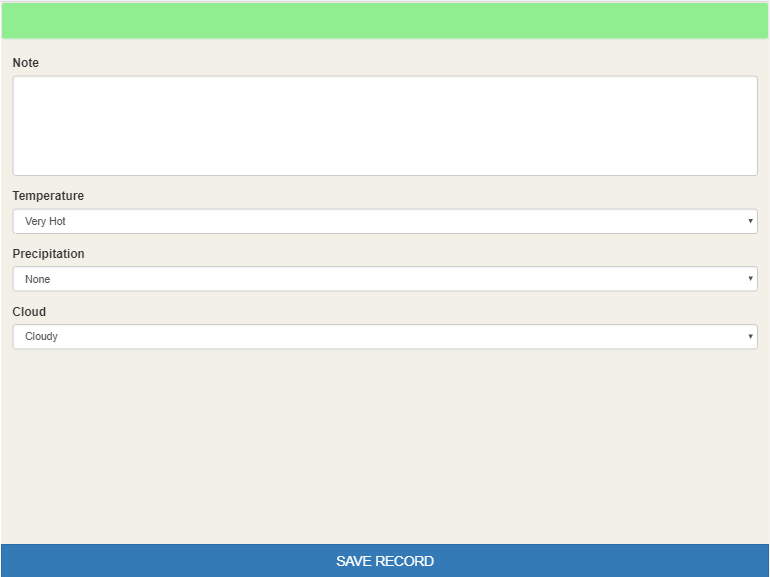


Figure 6: MDC Bow Hunting Mobile Application Interfaces

Survey development applications like SurveyMonkey, SurveyNuts, etc. [18] are readily available in the market for creating and deploying surveys for data collection. However, they come at a premium. These off-the-shelf products provide numerous features, which may be limited when creating a custom application. Firstly, many of these applications could cost from $10-$70 per month depending on certain user requirements, deployment platforms, and the application itself [18]. And the ones that come free of charge only provide limited features and only for a certain trial period. They also come with a restriction on customizability [19]. For instance, users may request changing the layout of an app but with restricted applications, that may not always be a possibility. As such, a tailor-made application like the proposed solution can provide more flexibility and control to the client over the design, usability and functionality within the application. Similarly, applications such as *SurveyMonkey* provide data analysis options with graphs and charts but do not readily provide a method of verifying the data as per our requirements. The lowest price plan in SurveyMonkey that generates graphs, charts and downloading responses and reports in CSV in called the Standard Plan, which costs $35 per month [20]. For our application, the MDC staffs require reports to be generated in a certain format. This will be done similar to SurveyMonkey in that these custom reports can be downloadable through CSV, while we also add the functionality of uploading data through CSV.

These works, including the previous MDC applications, were essential in identifying the existing approaches, and how they can be utilized for the proposed solution. They were used for helping build a well-rounded mobile and web application, which have been further explored in the following chapters.

# Design

This chapter discusses the design of the mobile and web dashboard for the MDC Creel application as well as the factors considered in its design. The chapter also covers the benefits and drawbacks of the development technologies used for the mobile application and web dashboard.

## Design Theory

Having a design plan for an application is essential in providing a user-friendly interface, especially for devices such as tablets and mobile phones, which have limited surface area. A well-designed application guides users through the application easily and with minimal assistance. To get maximum utilization from the user interface without overwhelming the user with too many buttons or options is essential for a high-quality user experience [21]. Creating such designs also helps in identifying the distribution of space on the screen, prioritizing the content, identifying any functions and how they will fit, as well as the relationships between the various interfaces [22], all of which are vital during the latter stages of the software lifecycle.

The design for the proposed mobile application focuses more on the users’ ability to navigate through the interface, collect data as efficiently as possible and provide an interface capable of uploading data to the remote server. The web dashboard focuses more on providing an interface to allow users to easily understand the data obtained through the survey – to be able to view, verify and in some instances, edit them through the web browsers. Similarly, having code that is reusable and well-organized is essential. This equates to writing fewer lines of code, essentially saving time, and enabling quicker addition of modules without disrupting any pre-existing code. This is more prominent on the web dashboard, which implements the object-oriented programming through PHP. However, it is limited on the mobile side due to lack of an existing modular framework at the time of design for the proposed Hybrid solution, but recently with the evolution of Single Page Applications (SPAs) a lot more Hybrid applications can be seen.

## System Architecture

### Creel Survey Mobile Application Architecture

The application’s initial design was based on the previous *MDC Shooting Range* application. MDC staff and clerks have been using it and recognize the swiftness, convenience, simplicity and flow of entering data with it. The proposed solution was designed with it as a foundation. However, the proposed solution has a larger undertaking, which includes dynamic generation of questions, simultaneous data entry for multiple users, and so on, and is designed to incorporate them.

#### Native vs. Hybrid Mobile Application

There are two major mobile Operating Systems (OS) in today’s market, Android and iOS [23], and each of them have their own process and frameworks for creating applications native to them, i.e. using programming languages such as Objective-C to create iOS applications or Java to develop applications for Android devices. Developing native applications provides numerous advantages such as better performance, access to all native APIs as well as providing a better integration between the native code and the device. However, developing native applications is costly, and depending on the requirements of the application, could take a long time to develop and deploy. This could be compounded if an application needs to be developed for both iOS and Android [24]. Similarly, there are applications available for mobile browsers, simulating web applications on the mobile devices. They are easier to develop and can be used on any mobile OS with a web browser but they do not have access to most native device functions (camera, GPS, accelerometers, etc.), and require internet connection all the time. The overall performance of the application is also low in such applications.

The proposed solution explores a different method of developing mobile applications – Hybrid mobile applications. Hybrid applications combine the simplicity of web based mobile applications and are able to simulate the performance of native applications. They are able to provide native-like functionalities with the help of plugins, which gives the application access to most, if not all, native device hardware and software features. The cost of developing using Hybrid applications is also cheaper than native application development, and utilizing the Hybrid approach allows a more rapid development and can implement applications quicker, and to multiple OS simultaneously.

|  |  |  |
| --- | --- | --- |
|  | Native | Hybrid |
| Development Speed | Slow | Faster than Native |
| Development Cost | High (licenses, devices, etc.) | Lower than Native (Open-source technologies) |
| Graphical Performance | High | Moderate |
| App Performance | Fast | Moderate |
| Access to Device Features? (Camera, Files, GPS, etc.) | Yes | Yes |
| Best suited for | Consumer-focused apps where performance and graphics necessary | Apps that do not have high performance requirements but need full device access |

Table 1: Native vs. Hybrid Mobile Application [24]

#### Phonegap and Apache Cordova

One of the most popular Hybrid mobile development framework is Phonegap [25], which has been used for the development of the *Creel Survey Mobile App*. This is an open-source distribution of Cordova framework and enables development of a mobile application using HTML5, CSS3 and Javascript. Utilizing APIs provided by Cordova, Phonegap enables interaction with the native device functions and is able to access native device functionalities. Phonegap encapsulates the application in a container (WebView) that is recognized by the mobile device’s OS and can deploy an application directly to the smartphone or tablet as if it were any other application downloaded from the online store. Phonegap applications are international standards compliant and will work on any mobile OS [26].

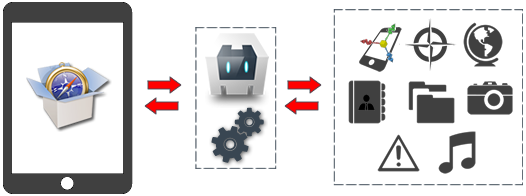


Figure 7: Phonegap APIs [27]

Phonegap

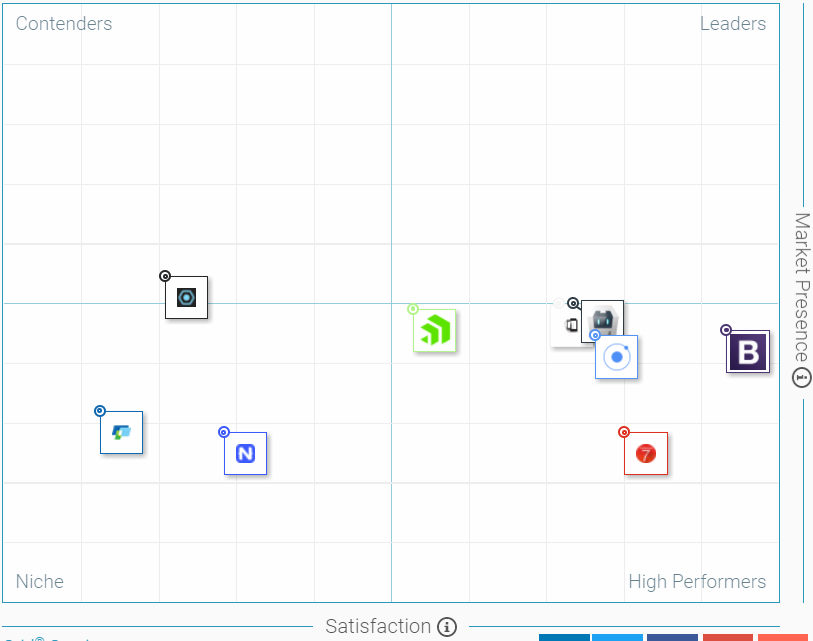
Cordova

Phonegap

Satisfaction

Market Presence

Figure 8: G2 Crowd Grid® for Mobile Development Frameworks [25]



Cordova enables communication between the rendered WebView, as seen in Figure 9, and the mobile OS with the help of plugins developed by a global community of Cordova developers. Cordova assists Phonegap with these APIs to communicate with the mobile OS sensors like GPS, Camera, Accelerometer, etc. Cordova additionally provides plugins for a persistent database storage with the help of SQLite, which allows CRUD (Create, Retrieve, Update, Delete) operations on data through SQL queries.

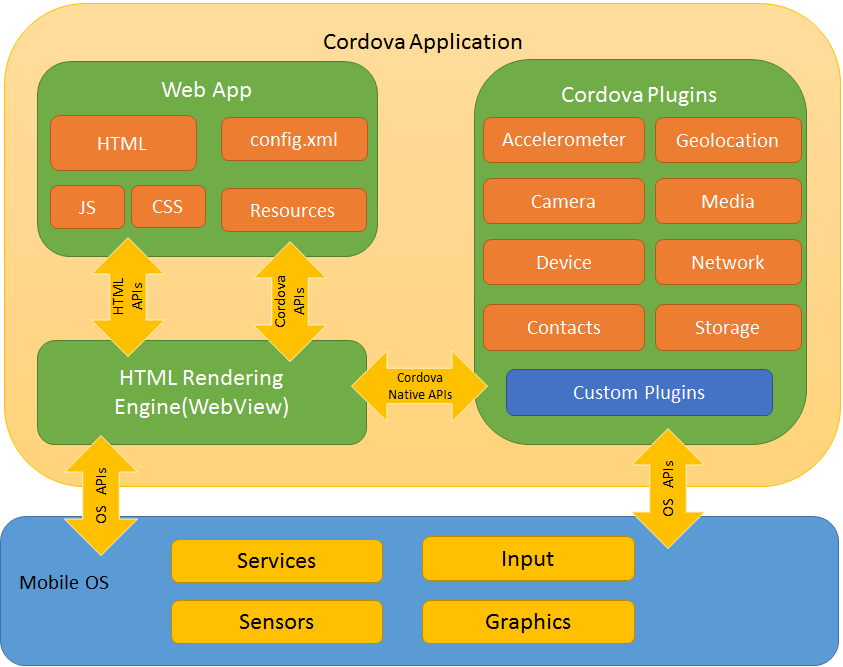


Figure 9: Cordova Architecture [12]

#### Useful Cordova Plugins

The Creel Survey Mobile Application requires access to many native device sensors and the following Cordova plugins were used for those purposes.

1. *Cordova-plugin-camera:* Provides access to the native camera functionality
2. *Cordova-plugin-network-information:* Allows access to the current network setting of the device, i.e. identify whether the device is connected to Wi-Fi, 4G connection, and so on
3. *Cordova-plugin-device:* Gets information of the device’s metadata such as the platform (OS), device’s UUID, manufacturer, serial number, etc.
4. *Cordova-plugin-dialogs:* Creates custom alert messages dialog boxes. This was especially useful for iOS to edit the alert titles
5. *Cordova-sqlite-storage:* Provides access to a persistent data storage option, which can be accessed using SQL queries
6. *Phonegap-plugin-barcodescanner:* QR code scanners were included in the initial design and development stages of the application

#### Design Considerations for Creel Survey Mobile Application

1. To make the best use of the **whitespaces** in tablets, which will run the device, the design of the application will be restricted to a landscape view rather than having both landscape and portrait. This will be useful in incorporating counters at the foot of the screen.
2. To maintain consistency with other iOS application alerts, Cordova’s **dialog plugins** will be used, which will ensure alerts replicate other iOS alerts.
3. Typing is difficult on tablets, and two-handed typing is almost impossible when holding the tablet and taking the surveys as is usually done. With that in mind, the input fields will try to include as much **alternatives to typing** (dropdown, toggles, checkboxes, etc.) as possible.
4. If an application places buttons at various locations in each view, the user will have a tough time getting used to the process – ensuring poor user satisfaction. Maintaining **consistency** in buttons, form inputs, validations, alerts, etc. makes for a better design concept.
5. The use of **HTML5’s LocalStorage** is vital when moving between the views. This preserves any necessary non-database details into variables for future use.
6. **Decoupling of functionalities** into multiple views, even when enough space is available on the screen, may be sometimes necessary. For instance, a request made by MDC during the requirements gathering phase of the project was to maintain two views for initiating a party and entering the number of anglers in the party. This could easily be done on the same page, but the purpose of the “New Party” page is to allow MDC clerks to remain idle on that page for a while and MDC staff prefer having to wait on that page until a party is identified for survey. Having multiple views decouples the functionality and prevents accidental button presses, which was a concern brought up by MDC.
7. Overwhelming the user with ***“Are you sure?”*** confirmation messages when moving between views. The initial design had a confirmation when moving between any two views but with the feedback obtained after testing, the alerts were limited to specific views, where this message was thought to be more necessary.

Moreover, the design for the proposed solution improves on content placement and organization, addition of new functionalities such as image capture, and will implement responsive design, i.e. capable of adjusting the screen to tablets and mobile devices of any resolution.

### Creel Survey Dashboard Architecture

The architecture of the Creel Survey Dashboard comprises of a web application hosted on the cloud service Amazon Web Service (AWS) EC2 server. The dashboard, which was developed using the dynamic website development tool LAMP stack, can be accessed through any web browser on mobile phones, desktops and tablets.

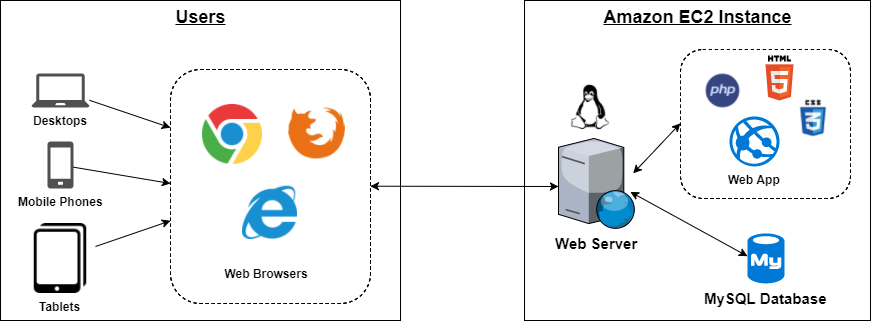


Figure 10: Creel Survey Dashboard Architecture Diagram

The acronym **LAMP** represents the following 4 open-source web development components: **Linux OS**, which is the operating system of the EC2 server, which houses the web application codes and the database; **Apache HTTP Server**, which processes all the HTTP requests and runs the web application on the EC2 instance; **MySQL**, which is a popular and free Relational Database and is used for storing the survey data; **PHP**, which is a popular server-side scripting tool used for web development, and enables communication between the front-end and back-end of the web application.

Prior to implementing PHP, the web pages of the dashboard had been created using HTML5, CSS3 using Bootstrap for a responsive design (resizes itself based on the display size of the device), and Javascript along with several jQuery libraries to make the interface more user-friendly. The object-oriented nature of PHP allows for an organized and modular approach to design and development. HTML code can be broken into modules and reused in multiple views. Furthermore, utilizing various in-built PHP functions such as the PHP Data Object (PDO) – the integration with the database and SQL commands is made more efficient, convenient and secure. And with the database residing on the same server, it gives the dashboard even quicker access to the data.

Finally, to implement modular, reusable and manageable code, the application must be categorized purposefully into a manageable framework. In accordance with object-oriented PHP, any redundant HTML code will be kept separately, resources such as images, custom CSS and Javascript will be separated, and so on. This will reduce redundancy and size of the application, improving the performance and allowing for a more rapid development of the application in the future. For the users, having the dashboard on the cloud adds portability to the application, allowing them access to the dashboard from anywhere.

## Data Design

Another major design aspect for the project was the preparation of database schema and the data itself. Data design was done in a top-down approach, first identifying the larger entities and schemas, and then breaking them down atomically. Similarly, it will be necessary to identify the most suitable way of transferring (download and upload) data between the Creel Survey Mobile Application and Creel Survey Dashboard, and the MySQL database.

### Creel Survey Mobile Application Data Design

Creel Surveys are primarily categorized into different *projects* based on the location, type of project, start and end times of the projects and so on. Creel survey data collection focuses mainly on the type of project, which are: Access, Stream, and Roving. Stream and Roving Creels pose a similar challenge in terms of data collection, which is why they are treated as a single entity in regard to their data handling.

Projects are the first entity to be assigned in the survey process. Without an existing project, surveys cannot be taken. Each project may have multiple schedules assigned to them, and each schedule may have one or more survey data. Every survey data represents a single interview with the angler party, and will consider the following information:

1. **Headers:** These include details such as the Clerk conducting the survey, details about the water conditions, any comments pertaining to the particular schedule, etc.
2. **Count times:** Every survey has a counter in its interface and depending on the type of survey, the counters will appear and disappear on the screen. Roving and Stream creels have specific count times and the counter will only appear during those times, while Access does not have a specific count time and will appear on the screen persistently.
3. **Counts:** These counts represent the values for the counters. The counter holds 3 values, which are interchangeable depending on the type of survey (access or stream/roving) for details such as angler party, non-angler party, etc.
4. **Parties:** Each schedule may have one or more parties and each party has one of more angler, each of whom have their own fish caught details.
   1. *Anglers:* Each party may have one or more anglers and their responses will be collected
   2. *Fish Caught [Images]:* Each angler will be surveyed about the fish that they caught during the scheduled time and optionally the images of the fish may be taken along with details about the fish, which will be used later during the verification process in the web dashboard

Each child node in Figure 11 will have information about their parent table (foreign key) to maintain relationship between the data. The projects, project species, schedules and count times will be downloaded from the server to carry out the surveys and the rest will store and upload data produced during the survey. All the data will all be written to an SQLite database.

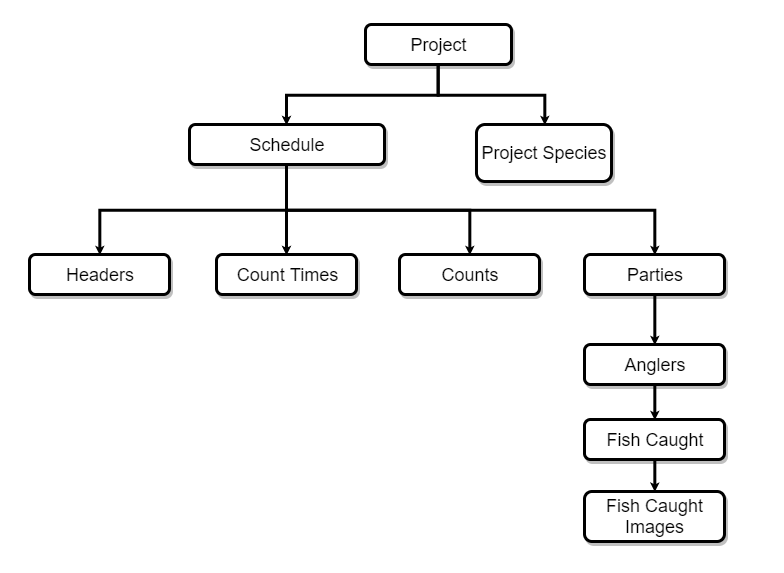


Figure 11: Creel Survey Mobile Application Data Design Tree

Another data design focused on is saving information when transitioning between the views. HTML5’s LocalStorage API will store persistent data in the form of a key-value pair. Until a party or a survey has completed, it will not be written into the database. Thus, to maintain the information when moving between views, HTML5’s LocalStorage will be very useful.

### Creel Survey Dashboard Data Design

The web dashboard will view, modify and verify all the data once it is uploaded to the database. Apart from the ones mentioned before in *3.3.1. Creel Survey Mobile Application Data Design*, the web dashboard will require interaction with a few more entities.

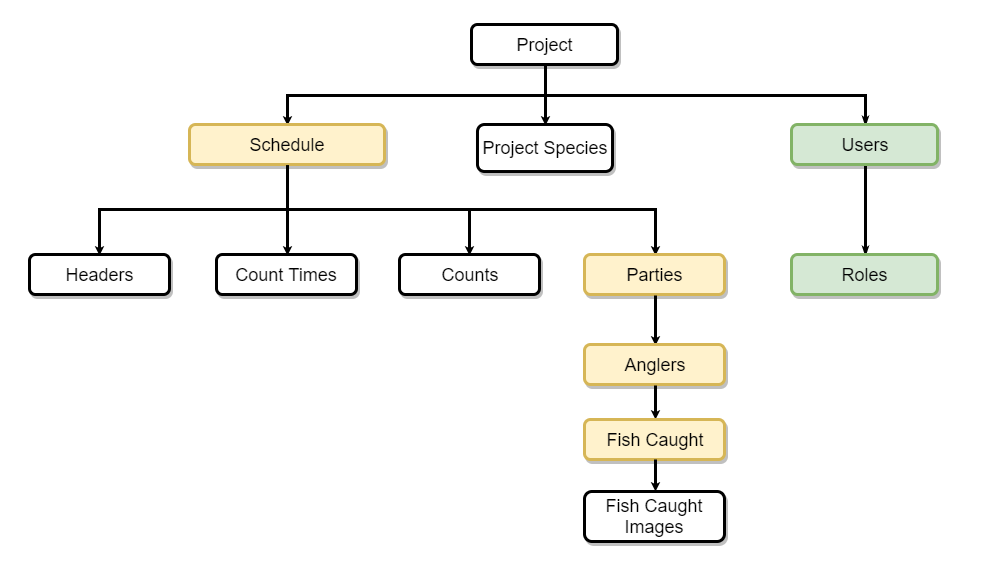


Figure 12: Creel Survey Dashboard Data Design Tree

In Figure 12, the yellow nodes (schedules, parties, anglers, fish caught) will be data that will need verification. The verification of fish caught will depend upon angler verification. The green nodes (users, roles) are new data that will be either created or modified through the web dashboard. Project management, user management, and data verification will be the major functions in the web dashboard.

## Technology Stack

There were multiple tools used for designing, developing and testing the application. They can be categorized into the front-end and client side, which were used for displaying the interface on the web browsers and the mobile app (Webview); the back-end and server side, which are the storage, and languages used to interact with the storage; and other utilities, which were the software programs and cloud services.

### Front-End and Client Side

* 1. HTML5: Markup Language on which the front-end interface is based for both the mobile application and the web dashboard. HTML5 also provides client-side storage through LocalStorage
  2. CSS3: Bootstrap CSS implements the responsive design on both the mobile application and the web dashboard
  3. Javascript and jQuery: Used for front-end validations in some instances and plugins the implement modals, DataTables, and so on

### Back-End and Server Side

1. SQLite: The SQL database engine for the mobile application
2. PHP: Scripting language used on the web dashboard and interaction between the mobile application and the web server
3. MySQL: Relational Database Management System on the web server to host all the data on the AWS EC2 instance
4. jQuery AJAX: jQuery Ajax is used for uploading and downloading the data to and from the EC2 instance via the mobile application
5. Amazon Web Services: This is the cloud service provided by Amazon, where the data for the web dashboard was hosted on an EC2 instance running a Linux OS

### Other utilities

1. Xcode: Apple’s iOS development and testing IDE
2. Sublime Text: A highly capable and feature-driven text editor
3. Git Bash: A bash emulation for Windows to connect to the EC2 instance; replication of the Linux terminal on windows
4. MySQL Workbench 6.3 CE: A very useful GUI to test SQL queries on the remotely hosted MySQL database
5. FileZilla: A secure and reliable FTP software to transfer files between the local system and the remote server and vice versa
6. Box: A file-sharing and collaboration service provided to Mizzou students and was used to back the code up regularly and securely

# Implementation

This chapter discusses the implementation of all the technology and requirements detailed in the previous chapters. The implementation details will be given in two parts: the implementation of the Creel Survey Mobile Application, its primary features and functionalities, and the Creel Survey Web Dashboard along with its features.

## Creel Survey Mobile Application

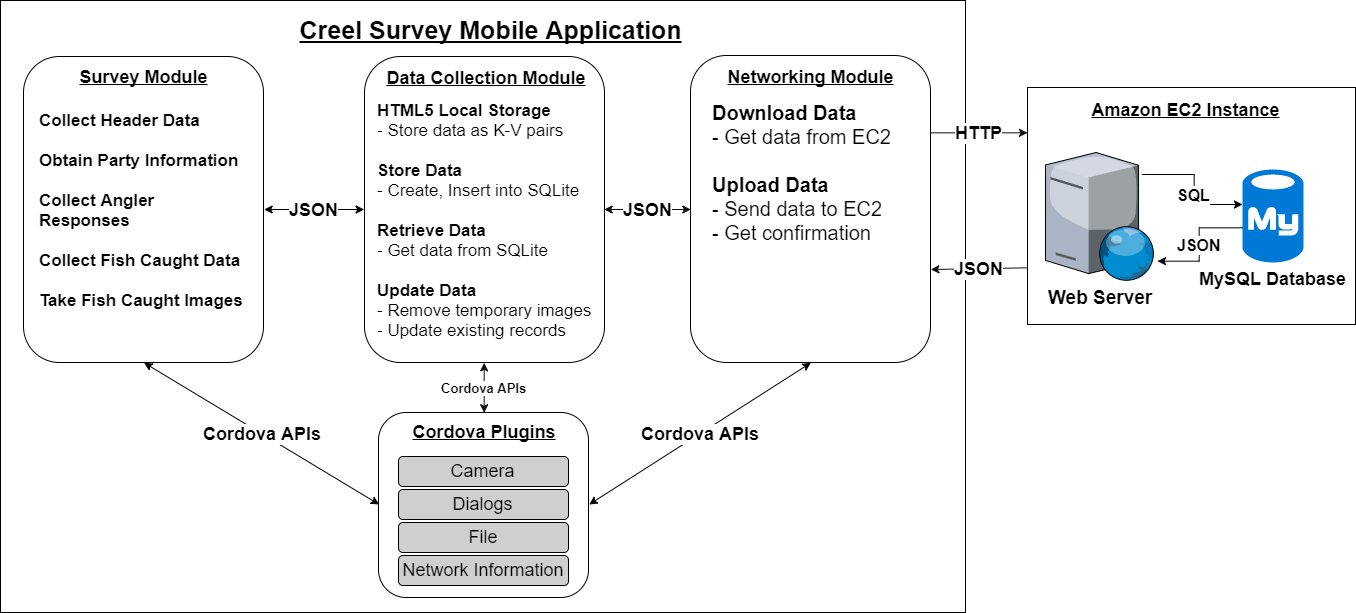


Figure 13: Creel Survey Mobile Application Implementation

The Creel Survey Mobile Application is a combination of the following components: Survey Module, Data Collection Module, and Networking Module. The *survey module* represents all the data related to the overall survey process, i.e. the surveys details, responses from the anglers, interaction with Cordova plugins to access the native device functionalities and so on. The *data collection module* represents the interaction with HTML5’s Local Storage API to store client-side data, the SQLite database that creates and inserts data to the SQLite database, and the retrieval of data to display on the application. Finally, the *networking module* represents the upload and download of data to and from the remote server.

### Survey Module

The survey module is composed of multiple views. Each view is represented by an HTML page in the code. In each view, there may be various components such as buttons, dropdowns, tables, etc. The purpose of the views is to allow users to carry out all the survey procedure in an efficient, user-friendly manner. Users login to the application, follow the procedure to conduct survey, collect all the necessary details and pass it onto the data collection module.

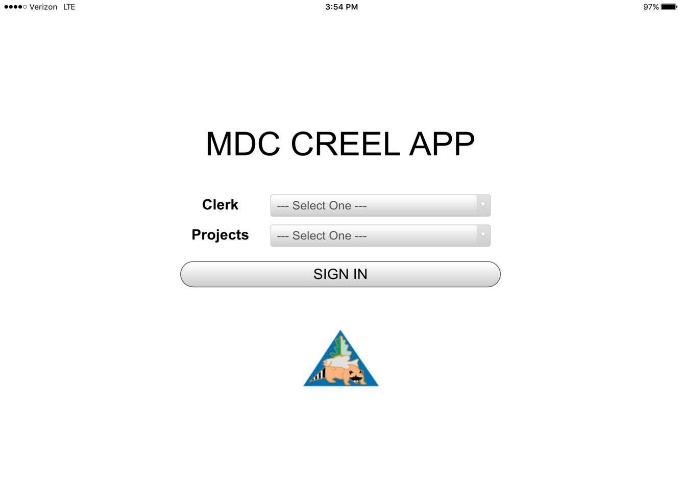
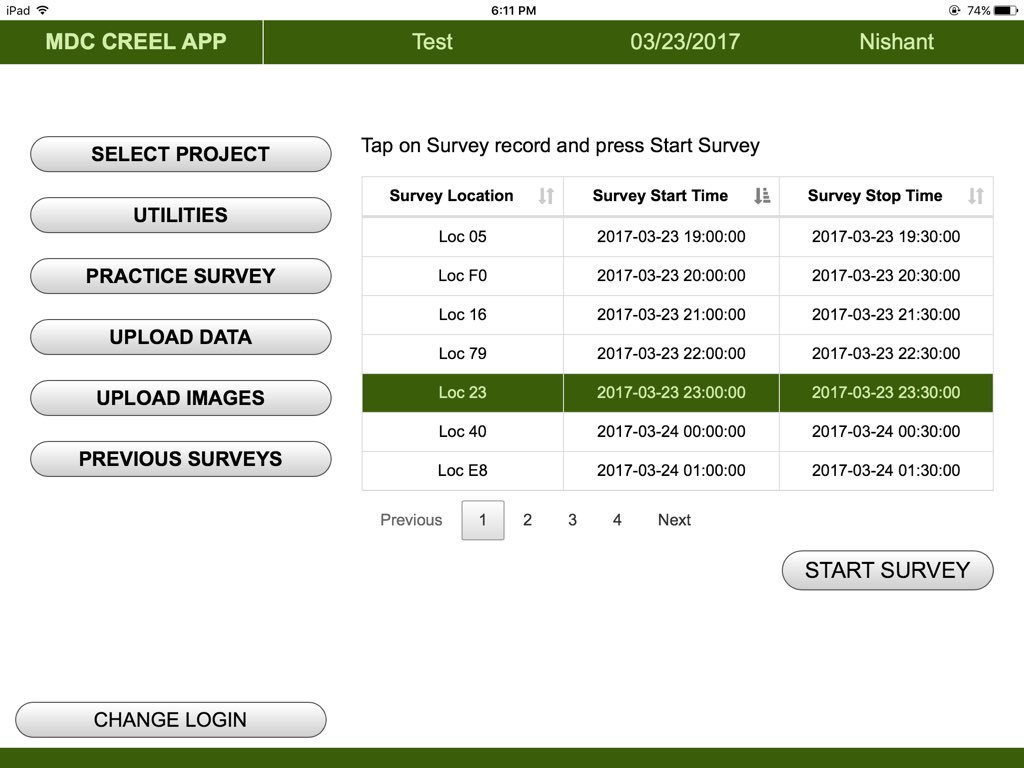


Figure 14: Creel Survey Mobile Application. Login; Homepage; Angler View; Tabular View

When logging into the application, the MDC Clerks enter their names and select the project they are working on. They are then directed to the main page, Figure 14a, and presented with a list of scheduled surveys under the selected project. Each survey on the list represents an individual record in the schedules table in the database. The clerk then selects a survey to begin the day’s data collection process.

When a new party is identified for the survey, the clerk initiates the New Party option. The clerks are also able to select one of two view options according to their preference. The clerks can either interview each angler sequentially, Figure 14c, or conduct simultaneous interviews, Figure 14d. When a new party is initiated, the clerk can then select the number of anglers in said party. If the party refuses to go forward with the survey then that will be recorded as their response.

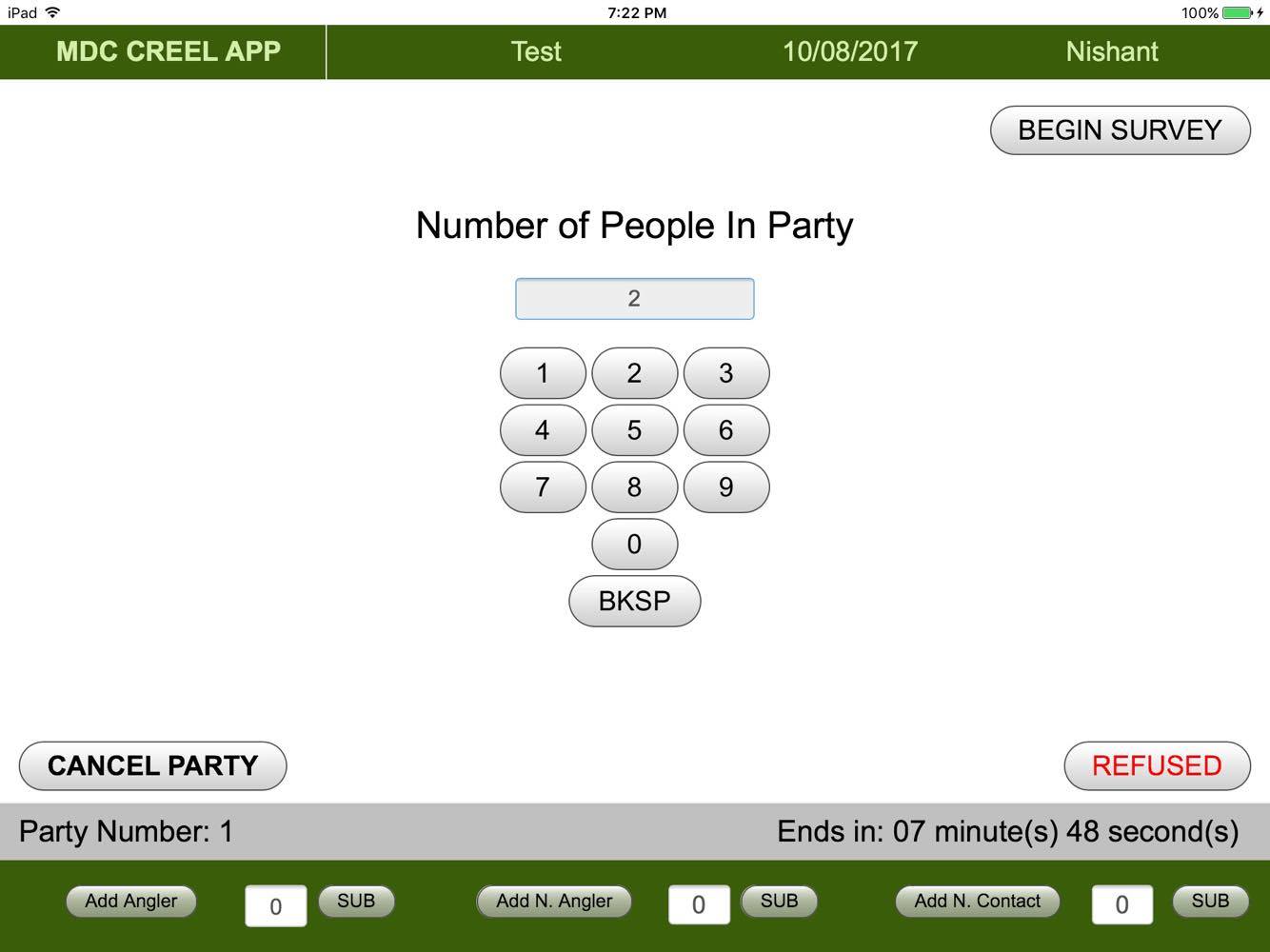


Figure 15: Party size initialization

When collecting the responses from the anglers, the clerks may also need to collect data regarding the fish caught by the anglers. This can be done by going to the Fish Caught page in the angler view, or selecting the fish caught button for the corresponding angler in Tab 2 of the tabular view. This will redirect the users to the page where for each angler, their fish caught details are recorded. In this view, the clerks will also be able to access the iPad’s camera using the Cordova plugin as in Figure 16, take a picture of the fish for verification on the web dashboard, and will be displayed as a thumbnail, as shown in Figure 17.

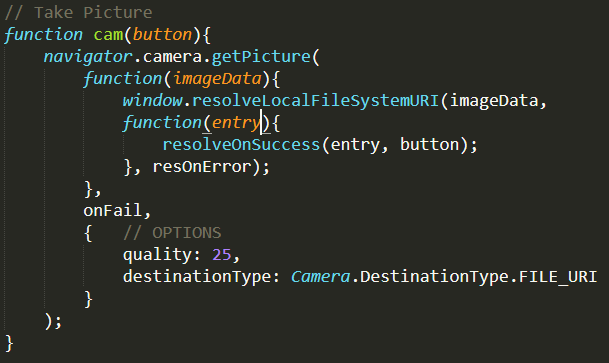


Figure 16: Cordova camera plugin code

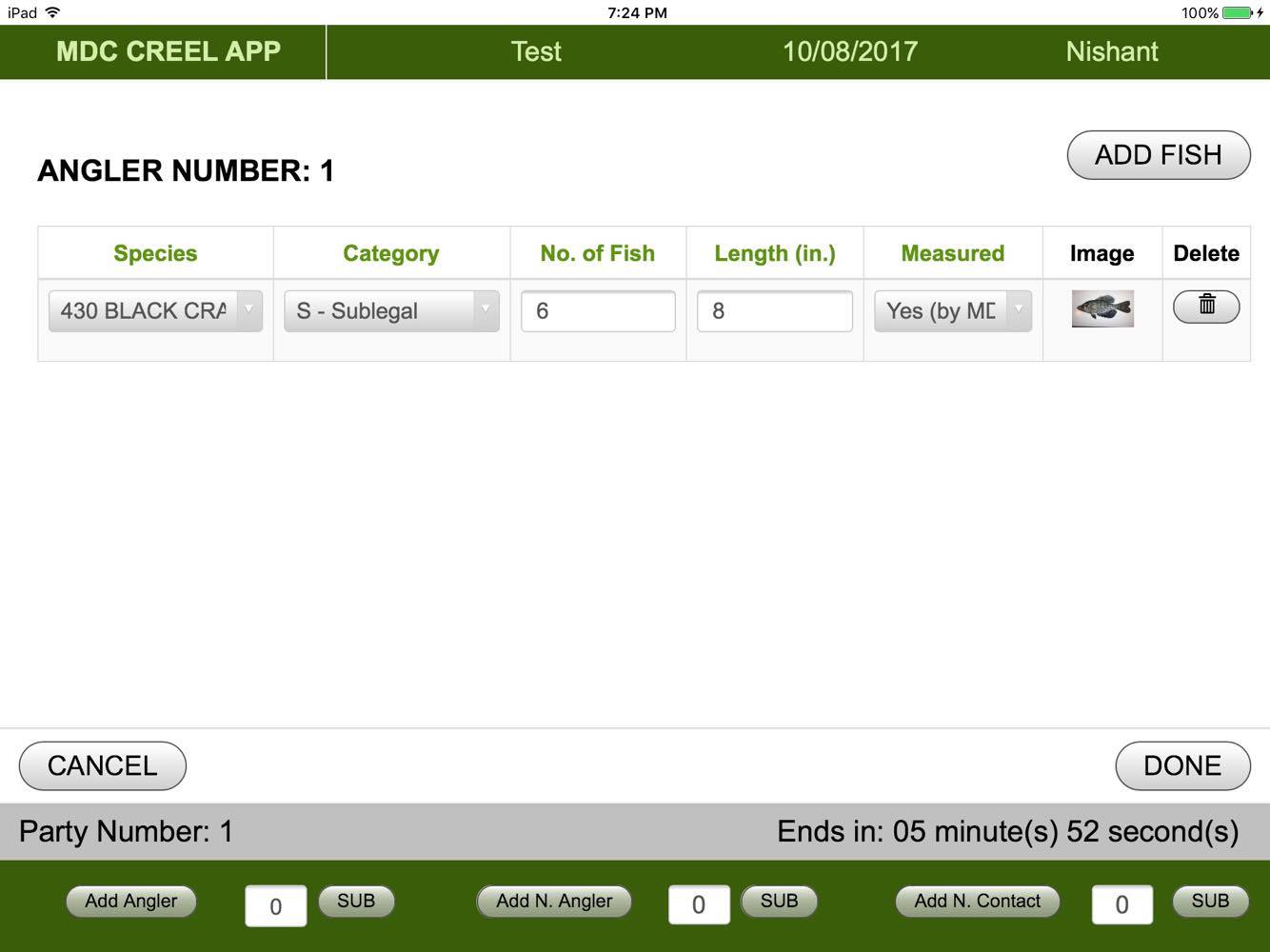


Figure 17: Fish caught view with fish image as thumbnail

After the angler details are collected, the data is sent to the Data Module to be stored in the database, which will be pushed to the database at the end of the survey by the clerks.

### Data Collection Module

#### HTML5’s Local Storage

Primarily when a survey is being conducted, the user moves between the different views, and if the data entered in each view is not preserved, it will be lost during the transition. And without completing the survey for each party, incomplete data cannot be sent to the database. To overcome this hurdle and prepare it for the database, the data between views is saved using HTML5’s Local Storage API.

This HTML5 Local Storage API is reminiscent of a key-value pair used in any programming language and most popularly used in web browser’s cookies. The key is given a variable name similar to any other Javascript variable while the value is in the form of a string. In some instances, the variable stores a single string value, e.g. to remember whether the “Tabular View” toggle is turned on or off or to remember the counter values in the interface, but in cases where the data is much more complex such as angler’s data and their corresponding fish caught data, then they are stored in a JSON array and converted to a string before setting it to a local storage variable. JSON arrays are extremely useful and convenient in storing these data. They are easy to maneuver and creating, modifying, and removing them is extremely convenient. Consequently, when retrieving the JSON array from the local storage, it must be parsed prior to manipulating the array data.

There is an exceptional case when it comes to saving the image data. The application is able to take pictures of the fish caught but needs to store that information as it will be uploaded later to the database. However, as cameras in tablets produce high quality images, the image size is quite large and storing it in the JSON array or even the SQLite database can lead to performance and memory issues causing the application to crash. To avoid these problems, the image is stored in the temporary directory of the application (inaccessible from outside the app), but its path is stored in the local storage before being sent to the database. When uploading the data to the server, the image is recursively retrieved via the path and the image is then sent.

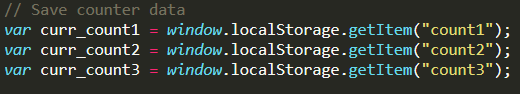


Figure 18: Single vs Nested JSON array stored in HTML5 local storage

#### Store, Retrieve, and Update Data

Once the survey is complete, the JSON arrays are written into the SQLite using dynamic SQL insert queries. Firstly, when each party is completed, the JSON array storing the entire party details, either through the angler view or tabular view, is retrieved, then parsed and inserted dynamically into the SQLite database per table, as shown in Figure 19. However, survey data is not written into the database until either the scheduled time has completed or until the clerk voluntarily finishes the survey. When the clerk completes the survey, the survey details (header, counts, and schedule) are inserted and updated where necessary.



Figure 19: Dynamically insert data into SQLite database

In some instances, data needs to be retrieved from the SQLite database, either to display them on the views as tables, as in Figure 14b, or to display as well as validate front-end information, as seen in the fish caught view. When adding information about fish, the clerk will need to enter details about the fish, for instance its length. Normally, fish of a particular species are found to be within a certain length (min and max length). This information is stored in the database, and when a clerk enters a value out of range, an alert will be displayed using Cordova’s dialog plugin asking whether the clerk entered the value accidentally. On confirmation, the value is either kept or cleared.

One of the significant issues discovered when retrieving data and displaying them with iOS WebView was the inconsistencies in handling datetime. For iOS, when manipulating the datetime with Javascript, there needs to be an uppercase ‘T’ between the date and time, e.g., “2017-10-25T10:00:00”, for the system to recognize that the time is actually 10 AM on October 25, 2017. To overcome this issue, custom functions were written in Javascript to convert datetime to an iOS recognizable datetime format. Moreover, there were similar issues when handling time zones on iOS, which was also tackled with custom functions.

Finally, when data are sent from the application to the server, the server sends a return message to the device with either a success or failure message. When data is successfully inserted into the remote server, the message directs the mobile application to update all the data sent from the device to reflect the successful upload, i.e. set the *upload status* of the data in the SQLite database to 1.

### Networking Module

The networking module is responsible for upload and download of the data to and from the tablet and the remote server.

#### Download Data from Web Server

Initially the details about the projects, scheduled surveys, details about the fish species and other pre-requisites for conducting a survey are loaded on to the database of the remote server, Figure 20b. At the first time of opening the application, there is an option in the interface that downloads these details onto the device. The download process involves accessing the device’s native network information using Cordova’s network information plugin to check whether the device is connected to Wi-Fi, Figure 20a. When a Wi-Fi connection is detected, the data is downloaded on to the device. The requests are sent synchronously to the remote server through jQuery Ajax in order to make sure all the data is retrieved. Once data for each table is downloaded, it is forwarded to the data collection module, which creates the necessary tables and inserts the records. The creation of the table is only done when new data is being downloaded and not each time the application is opened.

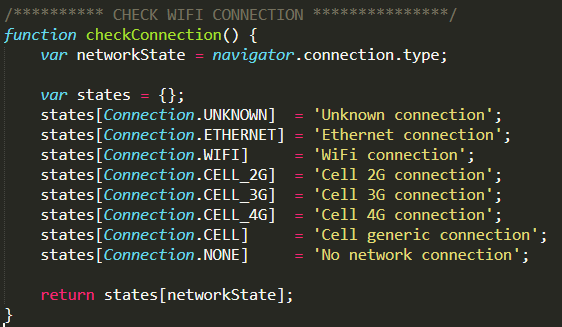


Figure 20: Cordova plugin to check network connection; Download data from server to device

#### Upload Data to Web Server

Networking module is also responsible for uploading the SQLite records to the remote server. To reduce the amount of data to be sent, saving time and bandwidth congestion, only necessary data is sent. The records, which have been produced by taking the survey, i.e. header, parties, anglers, and so on, are sent. The records are also checked whether they have been previously uploaded or not. Only records with an *upload status* of 0 are thusly retrieved from the database and prepared for upload. Similarly, to make sure cellular data is not used when sending enormous amounts of data, especially images, Cordova’s network information plugin is used to verify Wi-Fi connection.

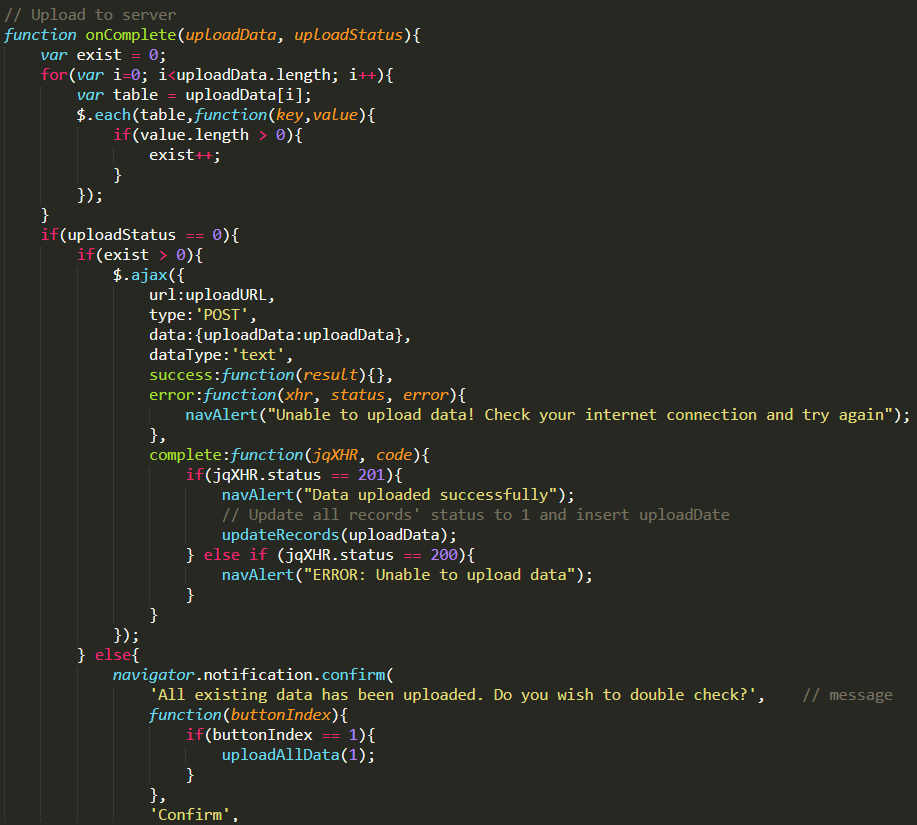


Figure 21: Upload data to server

There are two types of data that sent to the server: textual and image data. They are sent separately to avoid network congestion and also because the textual data takes priority over the images. Thus, the textual data is sent separately to ensure it reaches the database.

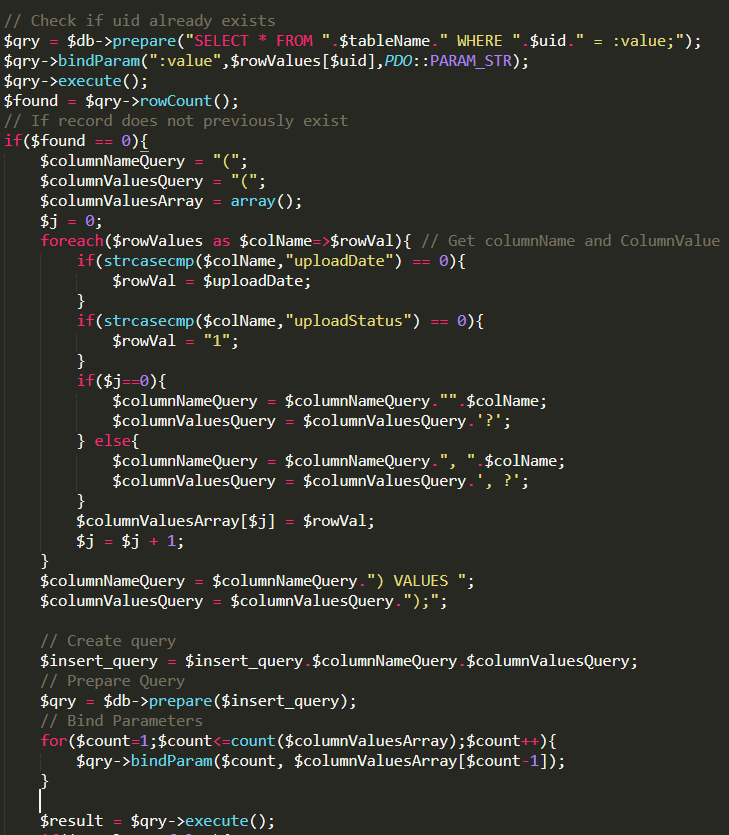


Figure 22: PHP Parsing uploaded data and executing insert queries

When a Wi-Fi connection is detected (or even a Cellular 4G connection for textual data), a function gets each SQLite table data as a single JSON array. The data for each table is then merged into a combined JSON array, where each table is identified by its own key. This combined JSON array is then sent as a single entity to the remote server’s PHP file, which parses the data, identifies each table information and inserts it into the corresponding table in the MySQL database on the server. The PHP file then returns a success or error message depending on the insertion, which is received by the Creel Survey Mobile Application. When the data has successfully been sent and stored in the server’s database, some of the data in the SQLite is updated to reflect the upload by the *data module*. Fields such as “uploadStatus” is updated from a 0 to 1, signifying successful insertion. When images are uploaded to the server successfully, they are removed from the application’s temporary directory, recovering any used memory for future image captures.

### Notable Features on Creel Survey Mobile Application

There are some notable features in the application, which are implemented in the survey module and data module using the Cordova plugins to provide a better overall user experience. Some of them have been listed below.

1. When entering data via a table, a **custom copy function** has been implemented on the table headers. Any header with a green-bold font is an interactive cell. Clicking on it will copy the column’s first cell’s information and cascade it onto the rest of the rows in that column. This is extremely useful and convenient when trying to enter the same data for multiple people

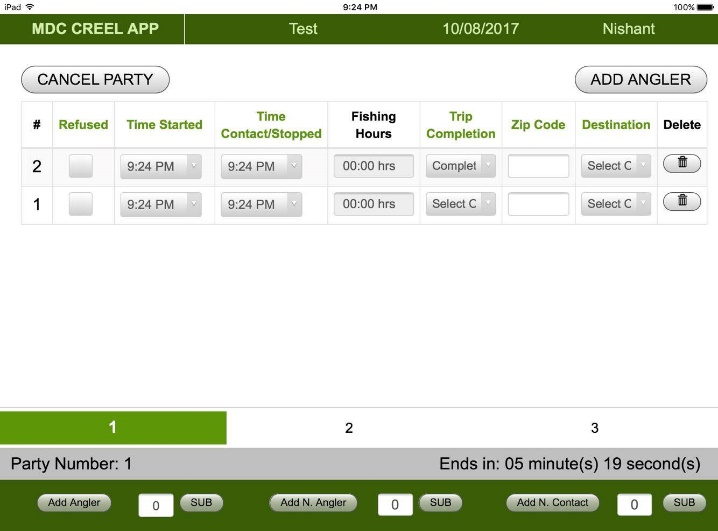
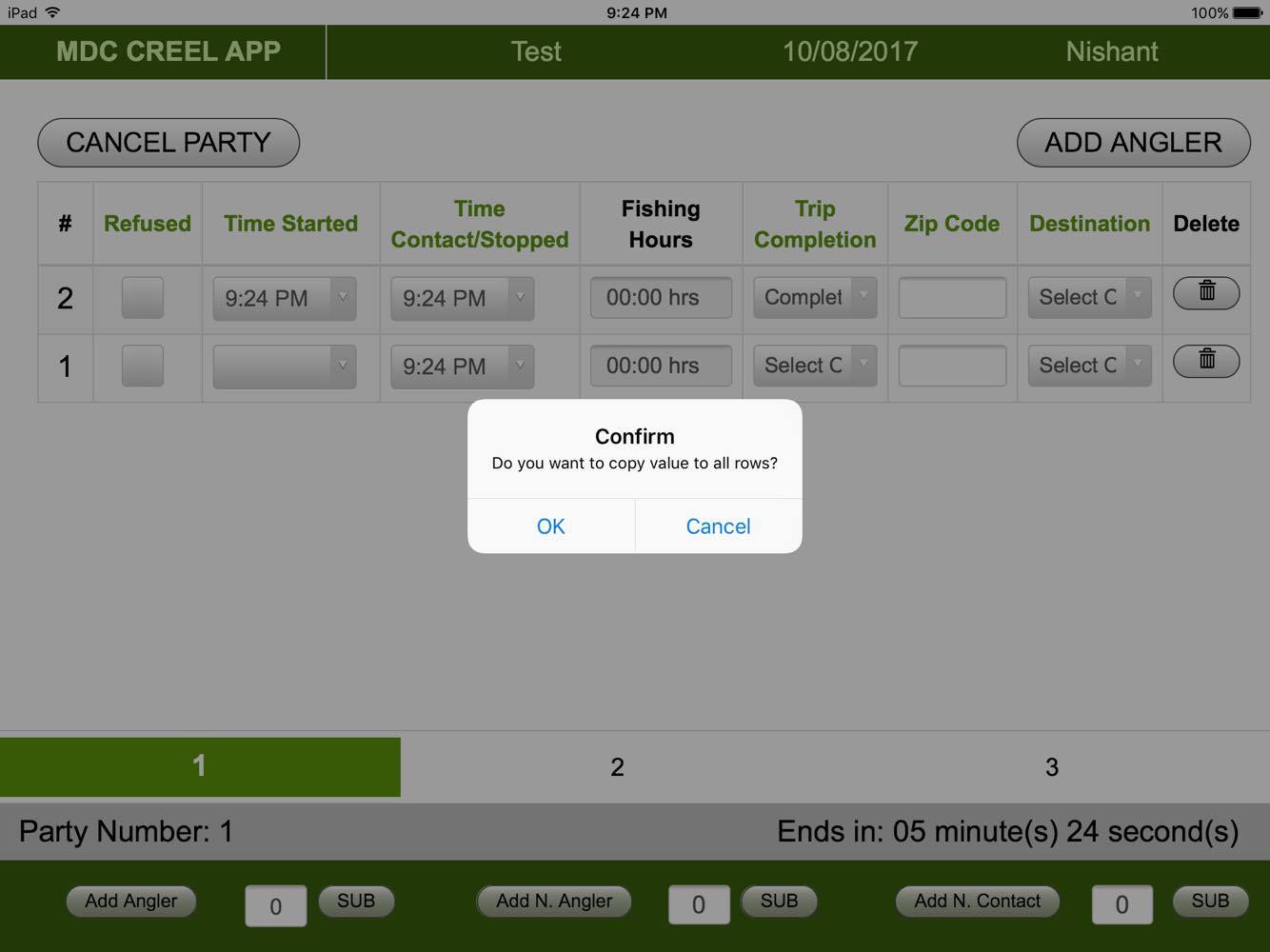


Figure 23: Copy-to-all function

1. In the Utilities view from the main page, there is an option to **view the database**. This view allows clerks to look at all the data produced through the surveys in the SQLite database and provides the clerks a way to ensure that all the survey responses have been stored properly, similar to Figure 53.
2. A single Javascript page has been implemented to **load all navbar and counter** information in every view with the help of Local Storage
3. Touch responses in the original implementation of the application was slow. Integration of a plugin **“Fastclick”** helped improve the responsiveness in iOS devices (was not required in Android devices). It was understood that iOS applications wait for 300 ms to check if the user intends to swipe, double tap or other actions. Using Fastclick helps identify and avoid any misunderstanding.
4. When downloading, uploading or interacting with the database, there may be some delays. To avoid users from interfering with the delay and making sure they realize a process is currently underway, **spinners** have been implemented to prevent users from accidentally moving to a different view, which could interrupt a significant process.



Figure 24: Spinner when loading

1. To train MDC staff, a **“Practice Survey”** button has been placed in the main page. This action will redirect the user to a survey process in the system. However, doing so ensures that no data will be saved to the database at the end of the survey.
2. Similarly, any incomplete or missed surveys by the clerks can be continued by going to the **“Previous Surveys”** view from the main page.

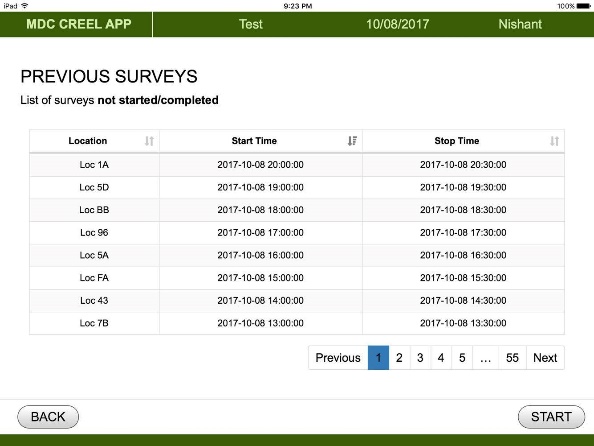
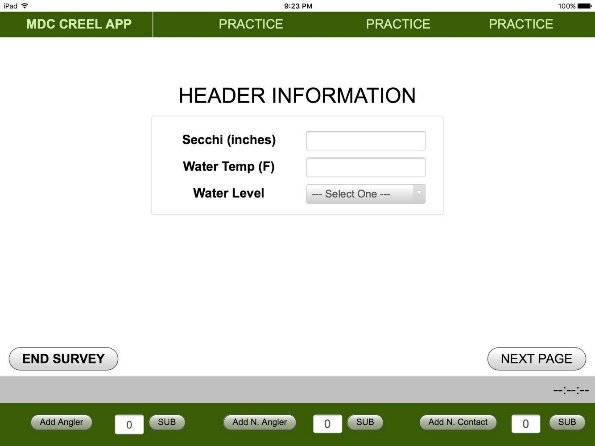


Figure 25: Practice and Previous survey view

1. Since the application is essentially a minified web application, it naturally follows the **web development architecture** for file organization, making it more structured, modular and reusable than the initial default structure, which is provided by Phonegap/Cordova.

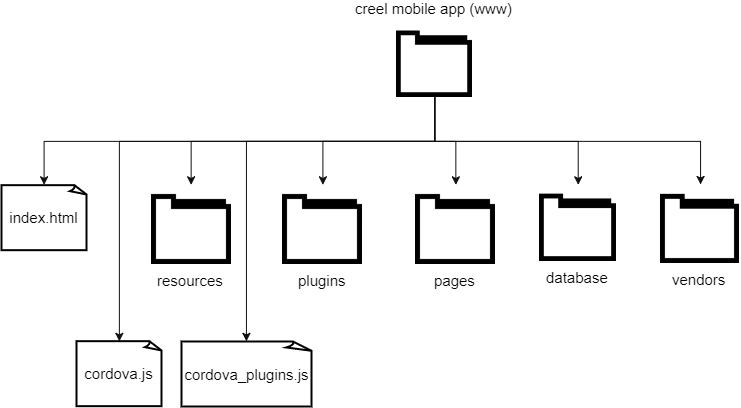


Figure 26: Creel Survey Mobile Application folder structure

## Creel Survey Web Dashboard

Creel Survey Web Dashboard allows MDC staff to view, verify and manage survey data through an interactive interface. Data, which has been uploaded to the server’s MySQL database, can be accessed through the dashboard in real-time. The interface is styled using Bootstrap CSS3 library to improve usability and allowing users a multi-platform access to the dashboard, i.e. mobile phones, desktops, by utilizing its innate responsiveness features. The major scripting language utilized to create a communication channel between the front-end and the back-end is PHP via MySQL’s PDO API.

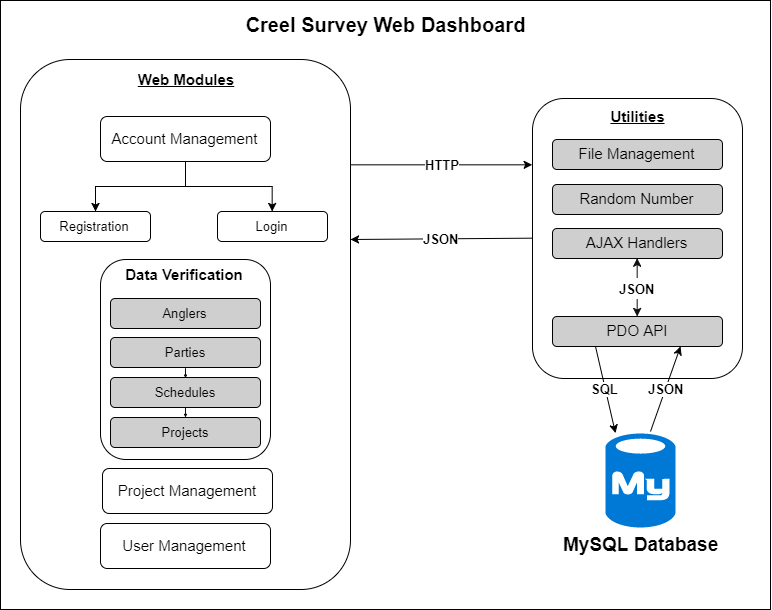


Figure 27: Creel Survey Web Dashboard Implementation

### Web Application File Hierarchy

Before writing any code, the framework for the code needed to be identified. This sets the standard for quality of the work, speed of development and efficiency of the entire implementation. Creating code that is reusable, compact and easy to follow will allows easier management of the application. Modifying existing code, addition of new dependencies and plugins, addition of newer modules are made feasible through a modular framework. There are many frameworks available but depending on the size, complexity and scope of the application, it may vary. For the Creel Survey Web Dashboard, the directory structure followed is as shown in Figure 28.

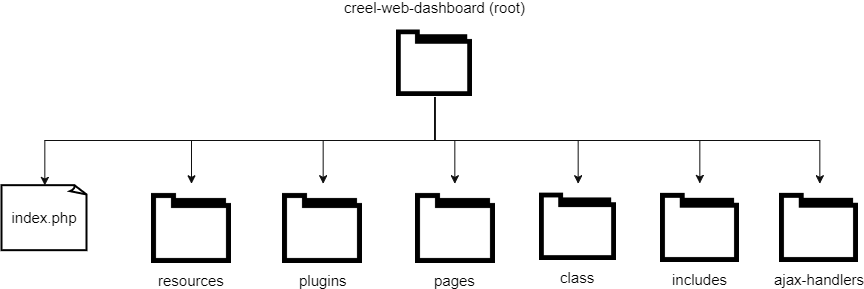


Figure 28: Web Application File Hierarchy

The initial page loaded by the Apache HTTP Server when the application is called by the web application is the *index.php* page in the root folder of the application. This page is kept for redirection to the login page inside the “pages” directory (*pages/index.php)*, where all the front-end views are stored. Any custom resources such as images, custom CSS styling, and Javascript code is stored separately in the “resources” directory. External dependencies and libraries like Bootstrap, Datatables, jQuery-UI and other CSS and Javascript libraries are stored separately in the “plugins” directory. To reduce the overall lines of code, all redundant HTML code, such as the navigation bar, footer, Javascript and CSS import codes are stored in their own separate PHP files in the “includes” folder. These files are then called using PHP’s *include\_once()* function whenever they are required.All codes for MySQL connection and PHP classes for each database table are stored inside the “class” directory. Classes and methods are created in table-specific PHP files. This enables the object-oriented implementation of PHP. SQL queries can be executed quickly, in a more manageable manner, and without having to re-write them on every page. Finally, in some instances, communication between the front-end and the back-end PHP code needs to be *controlled* through jQuery. jQuery provides more control of the data and avoids page reload when data is sent back and forth with the use of JSON arrays and Ajax. This communication is maintained through the PHP files inside the “ajax-handlers” directory.

### Account Management Module

Before being able to use the dashboard, users will need to register and login after getting authorization from the administrators. The first page that any user will come across in the website is the login page. For a first-time user, there will be an option to sign up, where they will be required to fill out some basic information (name, email address, username and password).

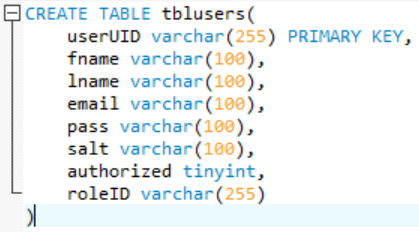


Figure 29: User and Roles table schema

The details are validated through HTML5 tags required and input types. After completing the registration, their names and other login details will be added to the database. However, they will not be able to sign in without authorization from the administrator. The authorization will be provided by the administrator through the *User Management* module, who will then assign the users their roles and inform the users via email.

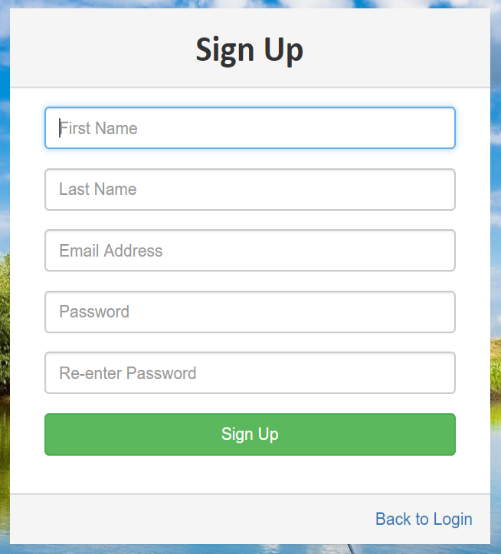
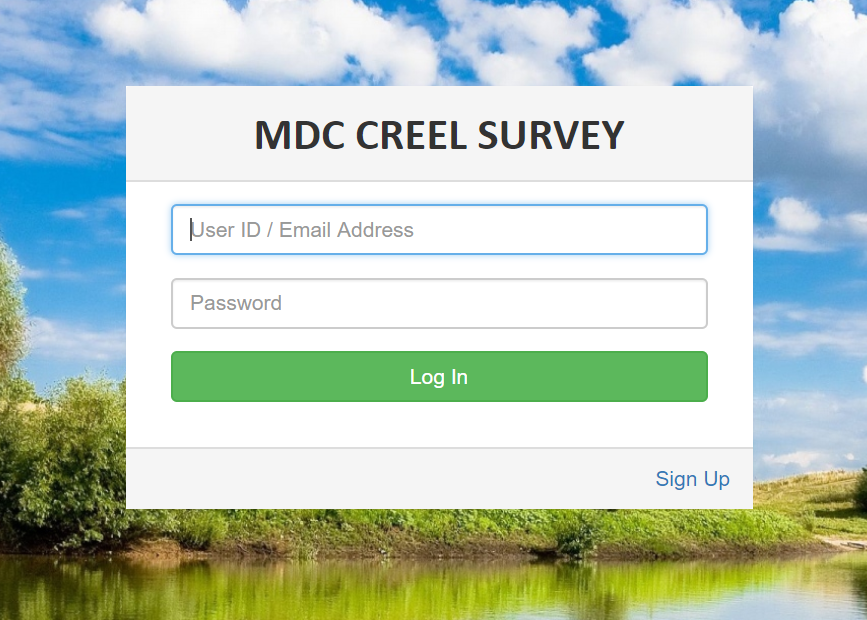


Figure 30: Login and Registration Interface

Security is paramount when creating accounts and creating a secure password prevents any unauthorized access to the application. Passwords are concatenated with a random string (salt) generated using PHP’s *mt\_rand()* function, which generates a random integer using the *Mersenne Twister Random Number Generator* and generates a random value quickly. The concatenated password is then hashed using SHA256, which encrypts the string. The purpose of salting a password prior to hashing it is to add a layer of randomization prior to encryption. This deters hacking into the system since the process of breaking it will be costly for anyone to accomplish.

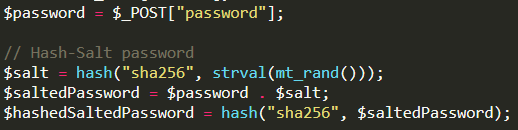


Figure 31: Salting and Hashing a password

### Data Verification Module

The web dashboard allows MDC staff to verify the collected survey data of its authenticity and whether it adheres to the expected responses or not. The mobile application has placed multiple validation methods to ensure only valid data are sent, but there may be some exceptional instances where data may be different than usual, e.g., an angler could have caught a fish with length longer or shorter than found for that particular species. Moreover, there are a lot of surveys conducted during every creel cycle and substantial amount of data is created in each survey. In order for those to be viewed, analyzed and verified conveniently and efficiently through the dashboard, the projects, schedules, parties, and angler data have been structured as a cascading parent-child hierarchy.

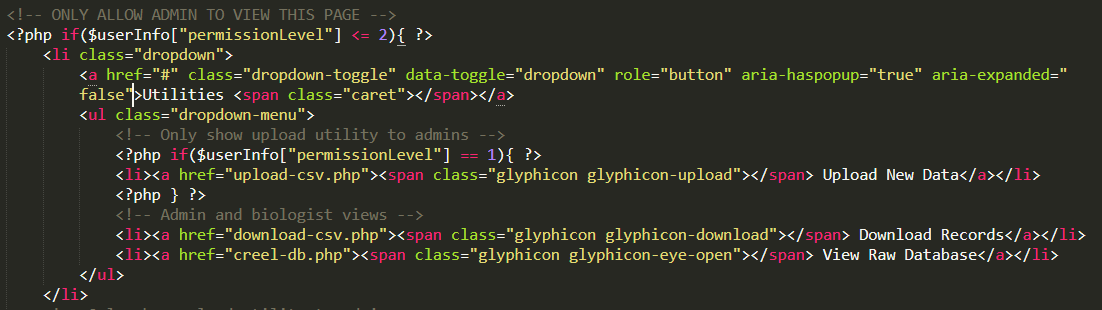


Figure 32: Administrator Permission Level validation

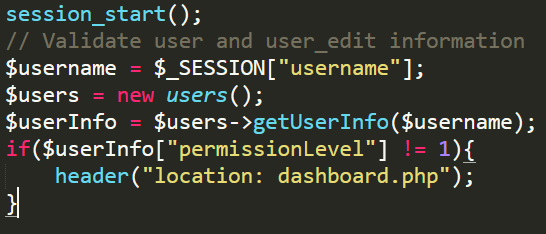


Figure 33: Unauthorized access redirection

After successfully logging into the application, the user’s username, user ID and permission levels are stored in separate PHP *$\_SESSION* variables. The application will present the user with a dashboard page, as shown in Figure 34,with a *treeview* of the projects on the left panel and an information panel on the right. At the top of all the views in the web dashboard there will be the application’s logo with a hyperlink to the homepage and a dynamic navigation bar directing the users to other available features on the application. The features available to the users through the navigation bar will depend on the permission level of the logged in user. Users will be able to either see dropdown items for User Management and Project Management (for administrators), or simply the utilities menu for the biologists and guest users. The permission levels will be validated on each page using the *$\_SESSION* variables to ensure only authorized users have access.

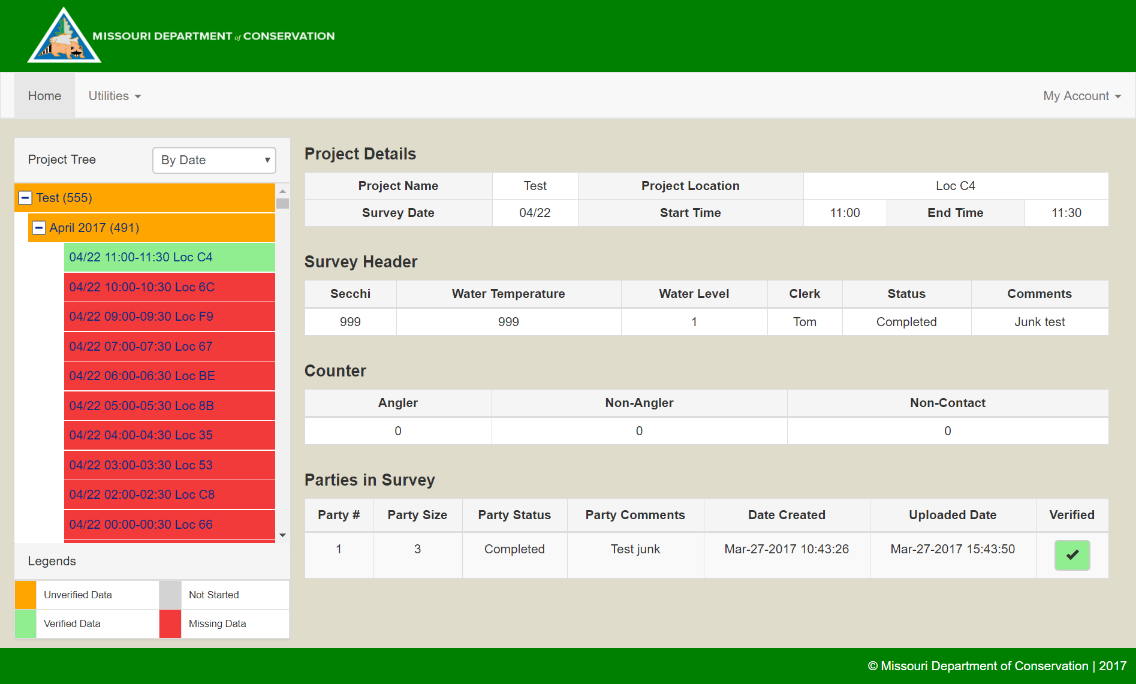


Figure 34: Web Dashboard Homepage

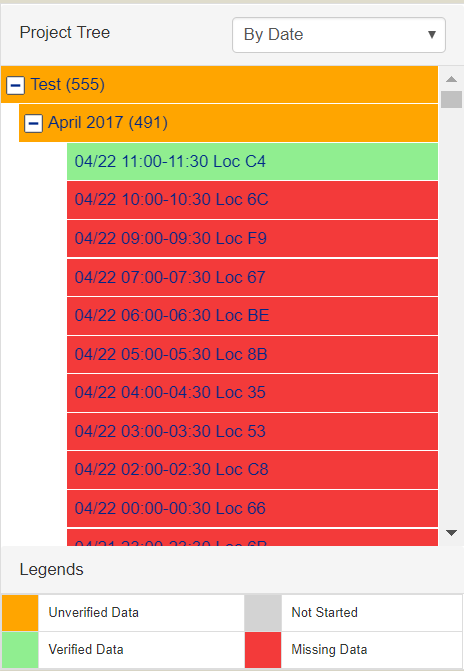
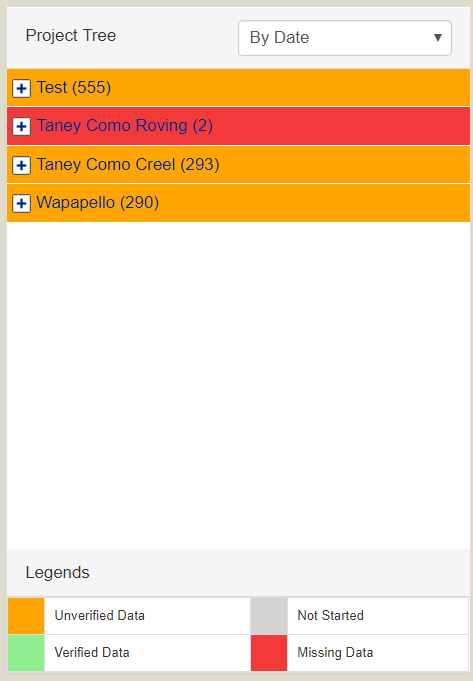


Figure 35: Web Dashboard Treeview and Information Panel]

In the main page, a hierarchical treeview presents the users with the list of projects. Each project can be expanded to present the months when surveys were or are scheduled. Further expanding those months will list each scheduled survey, sorted either per location or date, which can be changed to the user’s preference. The purpose of this treeview is to allow biologists and administrators, who are responsible for verifying the data, a more convenient way to sort through the surveys and any data (verified or unverified) present in the system. Furthermore, the treeview has been color coordinated to assist them in this process. Any surveys or projects, which have not yet started will have a gray background color. The surveys that have started and are unverified will be colored orange, and those that have already been verified will be colored green. Finally, any survey that has taken place, i.e. the survey date has passed, but has missing data in it will be colored red.

As more data gets stored in the database, loading the treeview can significantly reduce the performance of the application. It can take more than 5-10 seconds to load an entire view if the dataset is too large, as observed in [5]. To combat this problem, the parent-nodes of the tree are initially loaded with empty child-nodes. The child-nodes will be loaded only when required *(Lazy Loading)* [28]. Whenever the parent-node is clicked for expansion, its details are sent to the *schedule-handler.php* via jQuery Ajax in JSON arrays, which identifies project or month expansion request.

The handler then sends the request to the corresponding PHP function in the class directory and in return receives the corresponding details in JSON arrays, which is then finally returned to the treeview. This process modularizes, and significantly improves the performance and user satisfaction of the application since there is little to no wait time for the users. Furthermore, whenever a node (project or month) is retrieved from the database, the number of surveys in each node is obtained and placed in parentheses and the verification status is also retrieved.

The verification process has been modularized for convenience, usability and to ensure all the data, which requires verification has been viewed and confirmed. Displaying the entire survey data in the same view will cause confusion and may cause users to miss any incorrect data. Modularized approach allows biologists to correct any missing or incorrect data, which will prevent incorrect evaluation and analysis on the data later by administrators and other data analysts.

#### Party Verification

When a survey with data from the treeview is selected, its corresponding details and list of parties within those surveys are displayed in the information panel. There is a field in the party list, which shows whether the party is verified or not. The user can view the party details and its list of anglers by clicking on the verified/unverified button. The details required for party verification depends on whether or not all the anglers are verified within the party.

Biologists add a comment and verify the party by submitting the form. This sends an update request to the back-end through *parties-handler.php* via jQuery Ajax’s HTTP POST request through. The data sent also includes details such as the user carrying out the verification process obtained from the *$\_SESSION* variable, and the verification date and time, which are updated on the database. All the data is sent via JSON array to update the database records and a success or failure message is returned at the end, which displays the corresponding success or error messages on the front-end.

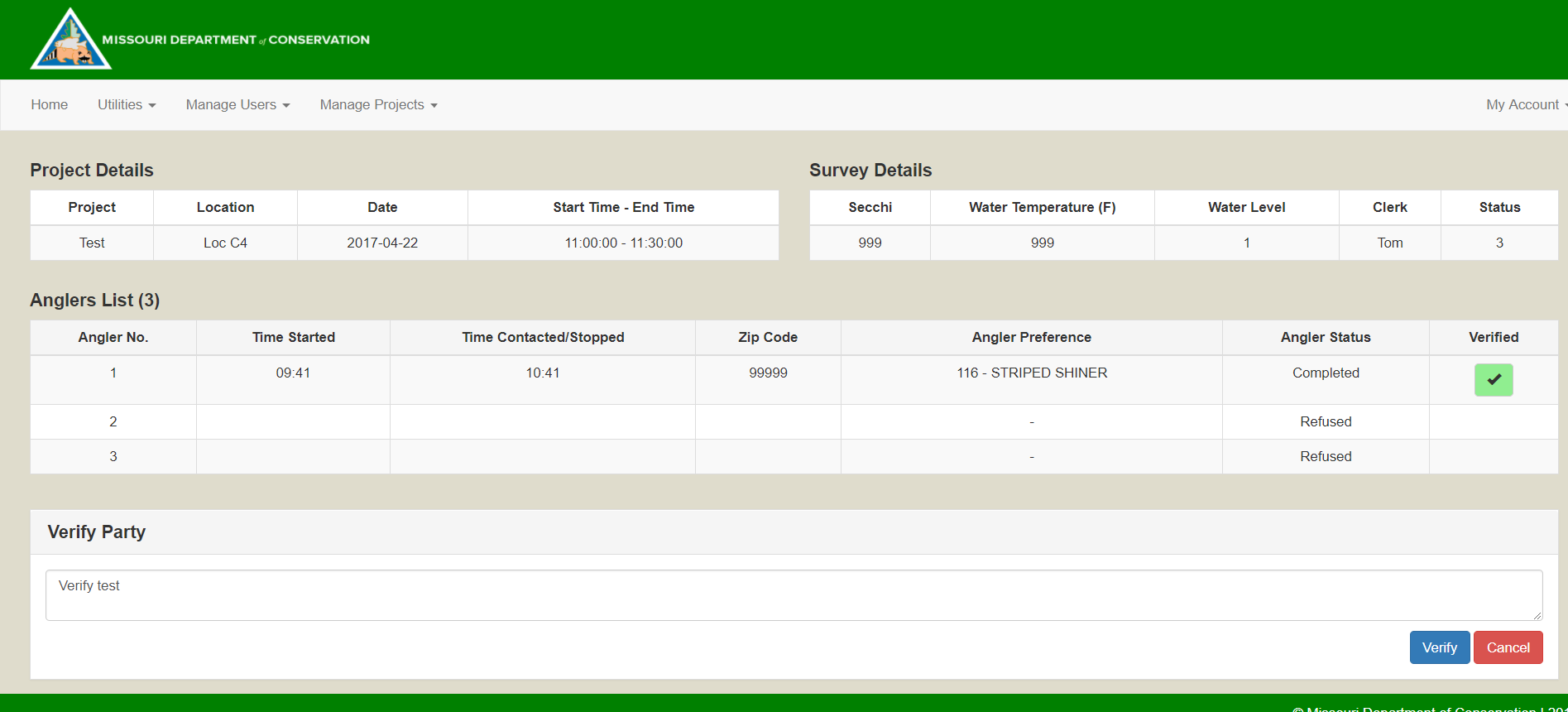


Figure 36: Party Verification Module

#### Angler Verification

Prior to verifying parties, all anglers in the should be verified. This is the leaf-node in the verification tree. The MDC staff will view the details of each angler, the fish caught and verify whether the information is valid, and in exceptional cases when the values diverge from the norm, whether those values are legitimate. MDC staff are experts in their fields and they are aware when values are unexpected, and in those cases, they must be allowed to change the values to an expected value or remove any invalid data from the survey. To allow administrators to change invalid and illegitimate data, but not interfere with valid data, text-fields in the view are displayed in text-fields, with the fish caught information having the option of being editable, as those hold higher priority, as shown in Figure 37. After all the data is verified, the user will add a comment and submit the form. This will send the entire updated information and comments to the database in JSON arrays through *angler-handler.php*, which again returns either a success or error message depending on the SQL process.

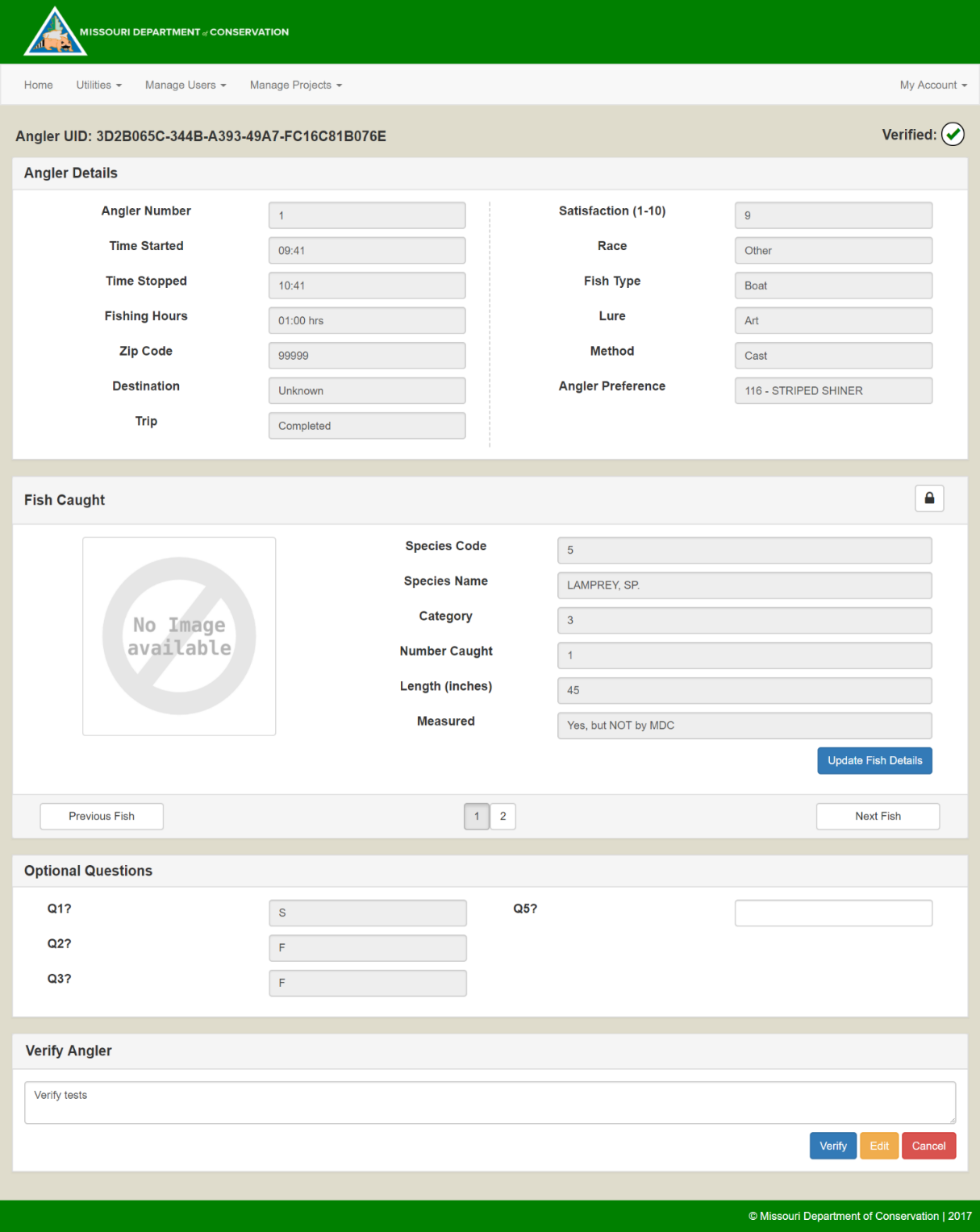


Figure 37: Angler Verification Module

Finally, once all the anglers and their parties are verified, the schedule they belong to are automatically verified on the front-end, and similarly when all the schedules are verified then the project they belong to are verified on the front-end. This can be observed only through the treeview. This process is referred to as *bubbling*, which is similar to the process of *cascading* but bubbling occurs in an upward direction (child affects parent).

### Project Management Module

The Creel Survey Web Dashboard enables administrators to manage the projects from the interface itself. Surveys cannot be added until projects already exist in the database. This prevents any unauthorized schedules from being listed in the database. The dashboard provides admins the ability to add and edit (up to an extent) any project on the database.

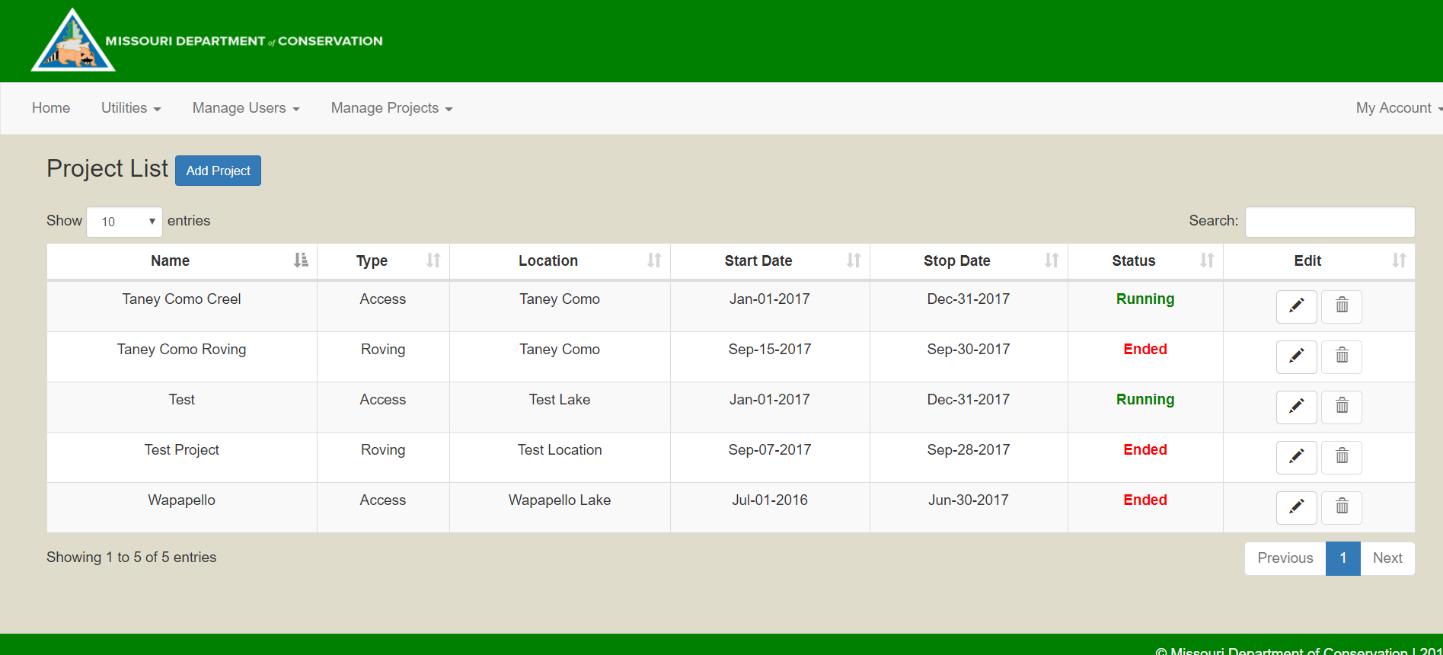


Figure 38: Project List View

The list of projects currently in the database can be viewed through the *Project List* item in the navbar. The list of projects, both currently running and ones that have ended can be viewed. Details about the projects can be seen in a tabular form. To view further details or to edit the details, the edit icon (pencil) can be clicked. These details are loaded using the PHP classes. The *getProjectList()* function in *class/projects.class.php* obtains the list of projects from the database and is inserted into the HTML table. The DataTable plugin is initialized after the data is loaded and this automatically enables sorting, searching and pagination functions on the view.

Details such as project name, location, type, start and stop date are required to add a new project, with the rest being optional data. For convenience, date pickers have been added to the interface preventing users from entering any incorrect date format, which is always a major issue. When a project is added or edit, its details are sent to the SQL queries through jQuery Ajax in JSON arrays. This prevents having to reload the page improving overall performance of the views.



Figure 39: PHP Code to insert data to the project-list table

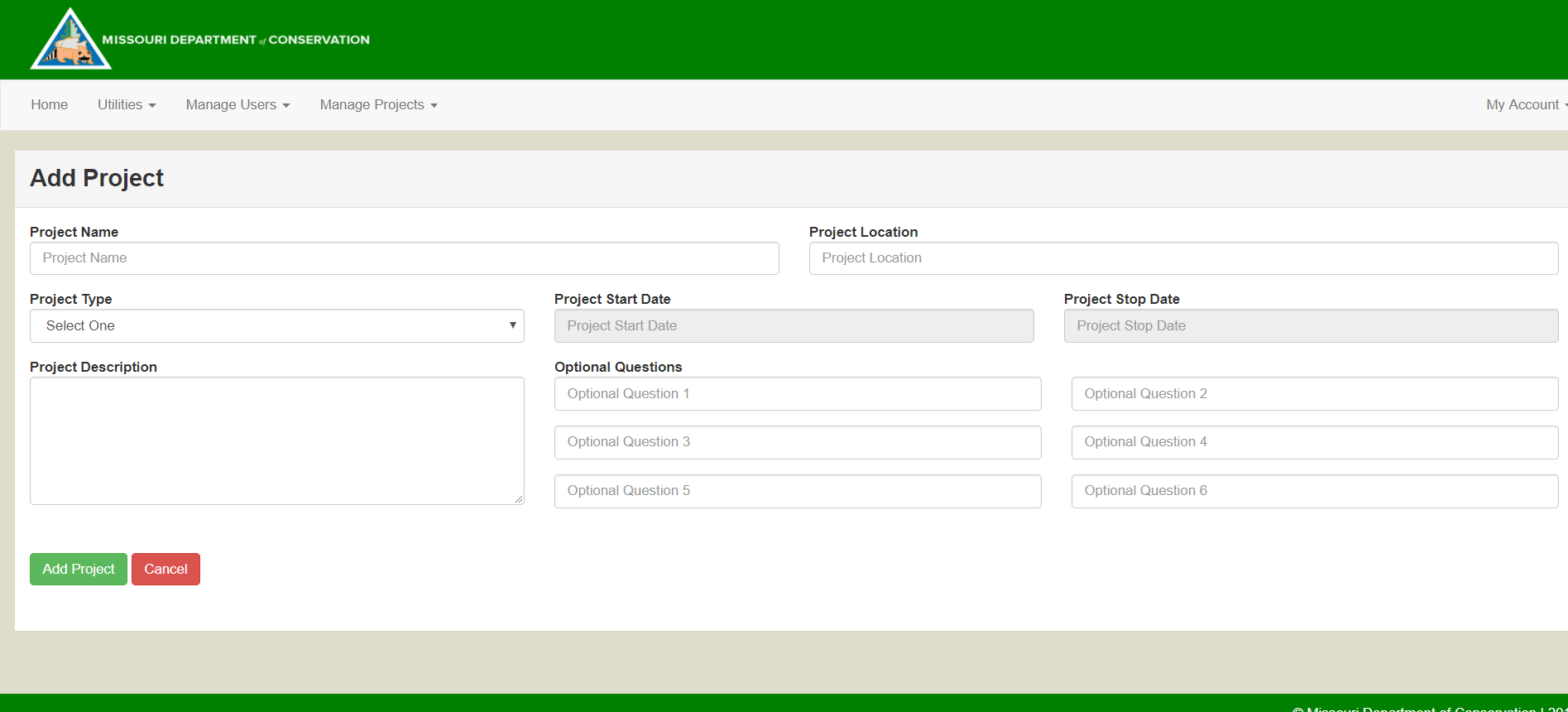


Figure 40: Add Project View

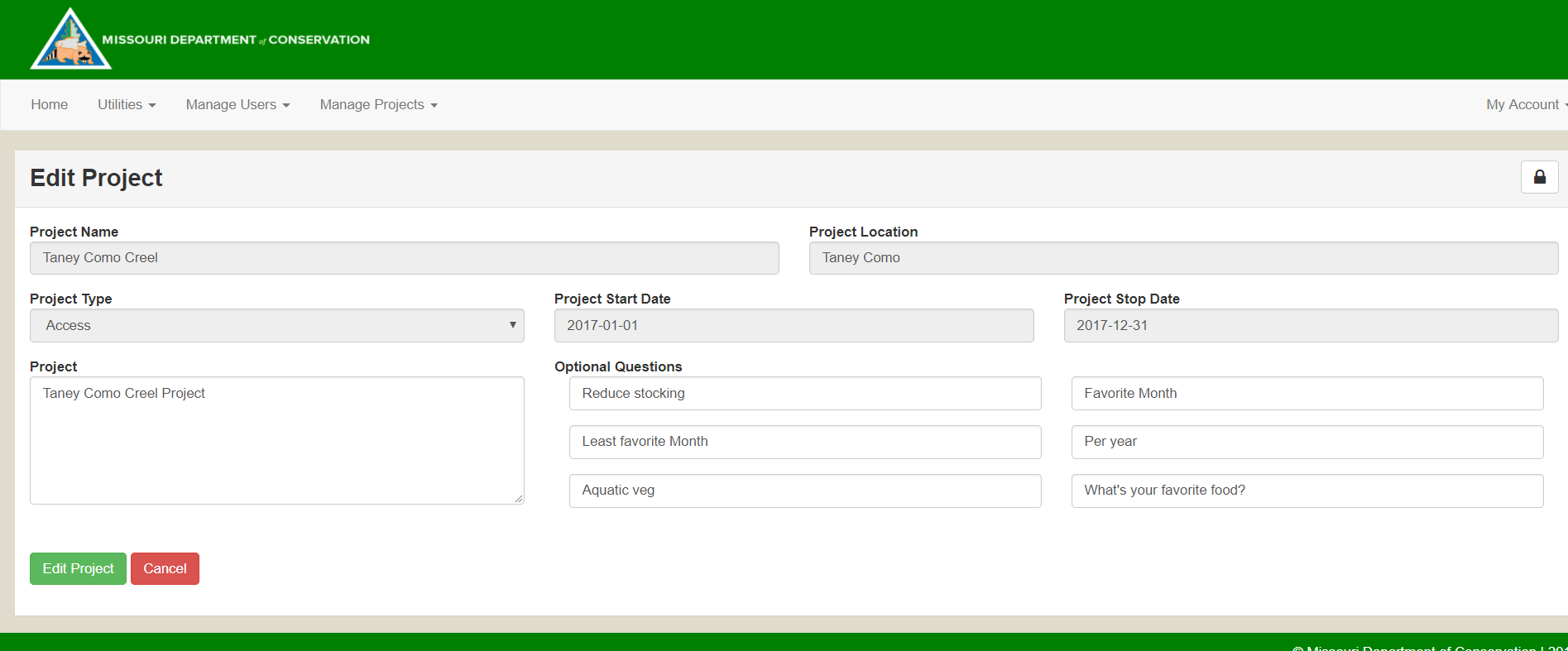


Figure 41: Edit Project View

### User Management Module

Similar to the project management, users (administrators, biologists and clerks) also must be added to the system. The user management module has been setup for this purpose. This module also has an interface to show all the users currently in the database. Administrators are the only users able to view, add and edit any user data. However, users are able to change their own profile information such as name, email and password. Users that have registered and are still *unauthorized* also appear in the user list interface, as shown in Figure 42, where authorization is given by an administrator.

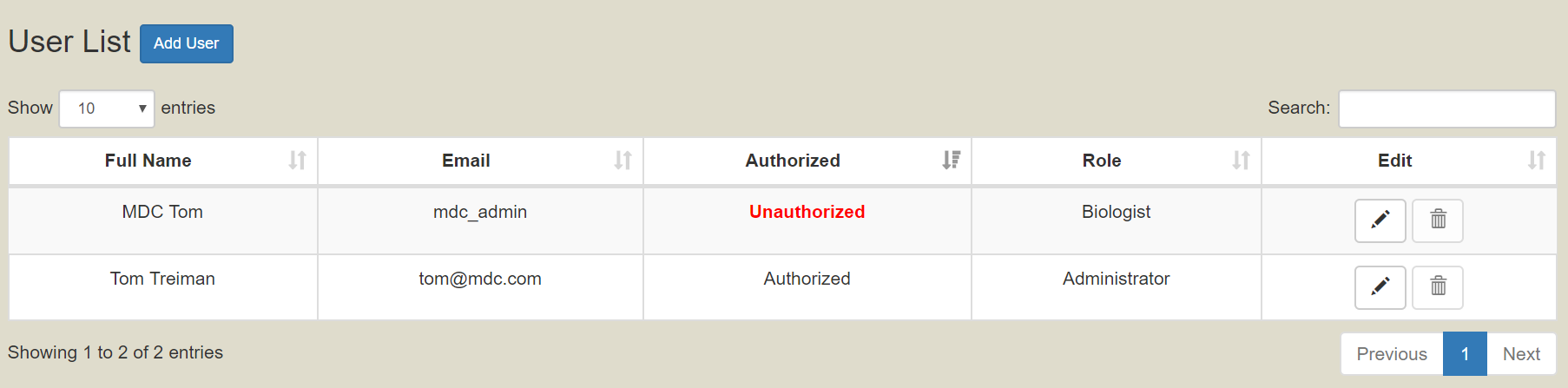


Figure 42: User List View

Administrators are able to add users, assign them roles and relevant projects. There are three main roles in the application: Administrator, Biologist, and Guest. The application is able to handle any new roles that are added in the future. Each role has its own purpose, with some having limitations as well.

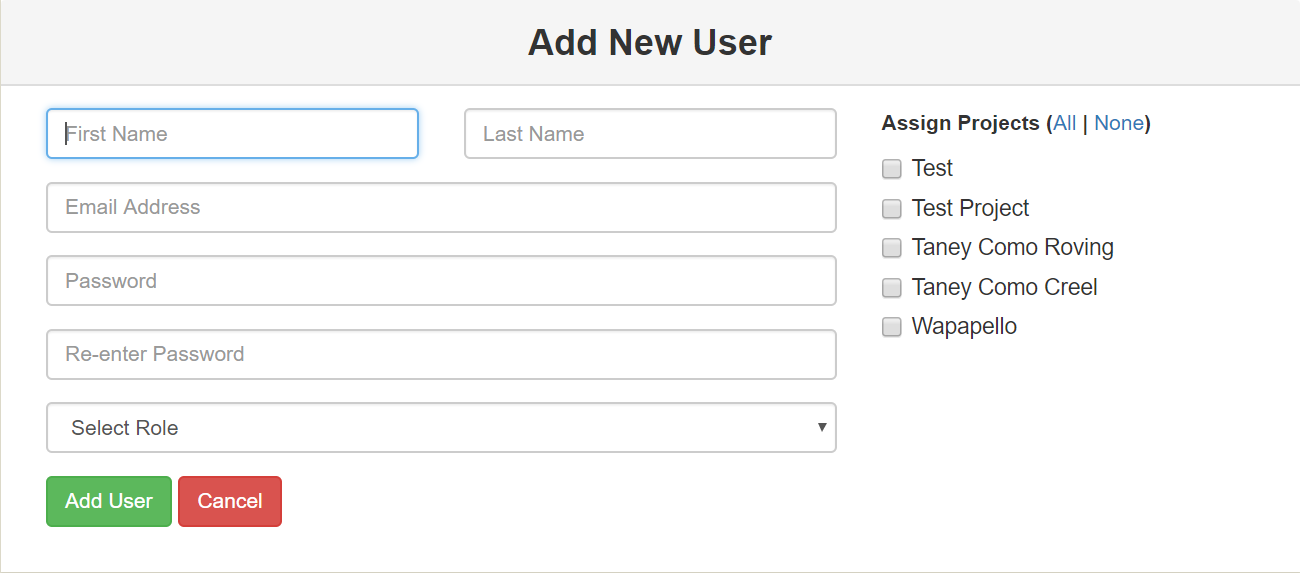


Figure 43: Add User View

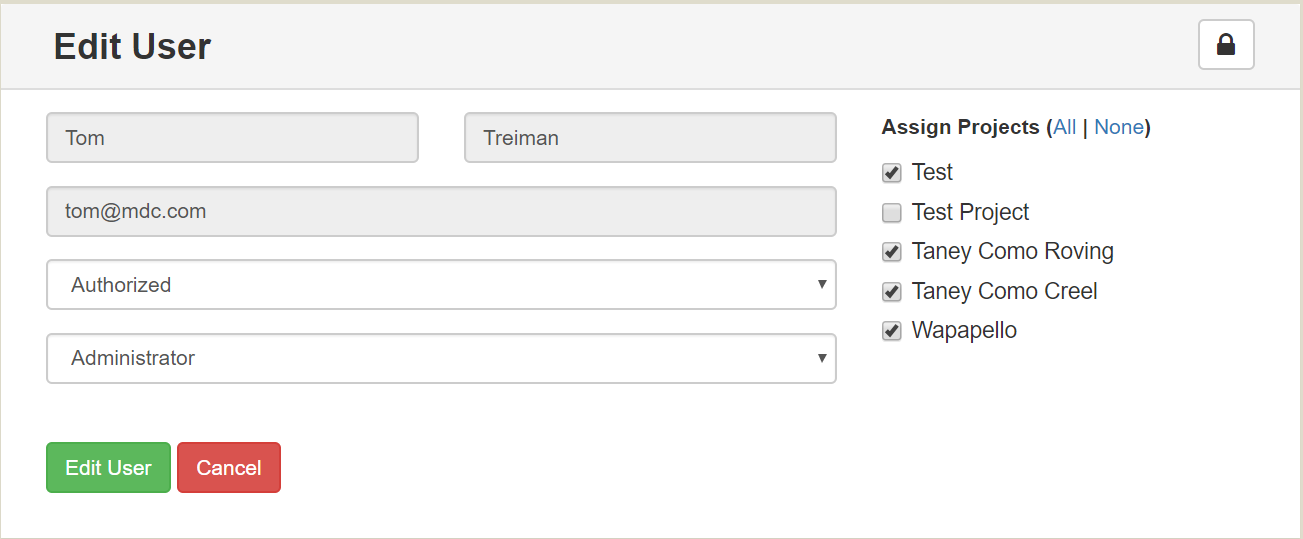


Figure 44: Edit User View

When a new user is to be added or an existing user is to be modified, their details are entered in by an administrator. User role is assigned or modified and depending on the role, the projects are assigned or modified. An administrator will have access to all project data, biologists are limited to a few projects at a time, whereas guests will be assigned no projects. This assignment is useful when downloading records from the database, which is elaborated in *4.2.6.1* *File Management*.

When a new user is added or an existing user’s details are updated, the information is sent to the database via PHP. However, in this stage two tables will need to be updated sequentially. Firstly, an SQL transaction will be initiated. This prevents any SQL query from auto-commiting to the database, ensuring the option of rolling back in case of failure. The user information will initially be updated on the user table. Then the user UID and all the projects assigned to the user will be inserted into the *project-user* associative entity, as shown in Figure 45. Similarly, if a new user is being added, then all the project-user relationships in the associative entity are first deleted and then new records are inserted to ensure there is no redundancy or failures.

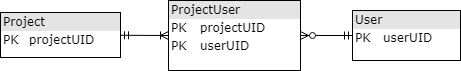


Figure 45: Project-User Associative ERD

### Utilities

Custom functions, classes and other APIs have been used in this application to make many tasks easier for the user. Permitted users are able to download and upload files, add and modify records by communicating with the database and so on.

#### File Management

One of the most useful aspect of the application is allowing administrators and clerks to download records to their local system from the database in a Comma-Separated Value (CSV) file. Administrators are able to add new records to the database by uploading new files in CSV format as well. This is extremely important because MDC produces data for tables like schedules and counts using data analytics software in CSV files. And these files can have hundreds to thousands of rows in them. It would be inconvenient, unfeasible, and even impractical to ask users to enter those records to the database manually. Therefore, add such data to the database, the web application has implemented a File Management module, which can help upload these files into the MySQL database.

Since projects can already be added by administrators, the most important records, which need to be added to the database are schedules, count times for the schedules and fish species specific to a project. In the utilities tab, of the navigation bar the upload and download data options are given.

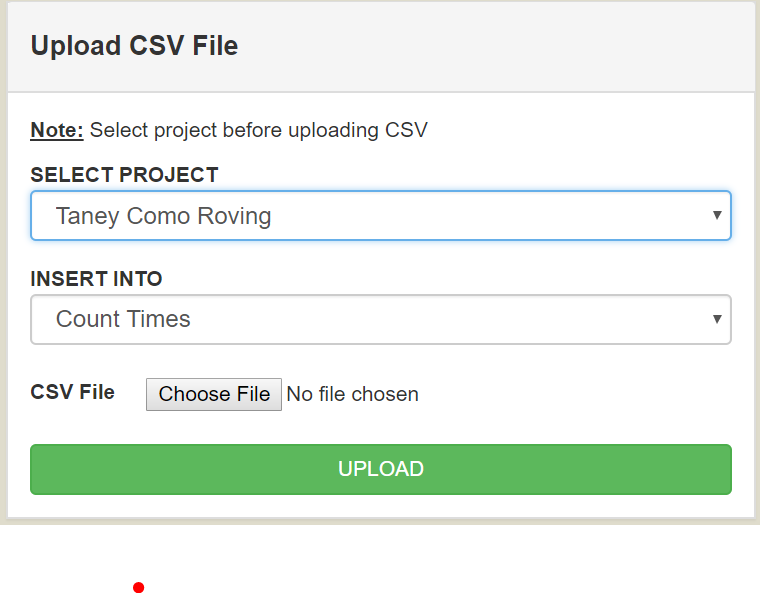


Figure 46: Upload CSV Interface

When administrators upload new records, the first step is selecting a project, to which the new records are going to be assigned. After selecting a project, the table which they want to insert the data into are shown in a dropdown list. The administrator then selects the CSV file they would like to upload and the code in the backend verifies that the data in the CSV matches the required data.

Firstly, the code checks that the number of columns in the CSV file matches the number of columns in the database, similarly when that is confirmed the next verification process checks whether each of the column names in the CSV files (first row of the file) and the columns names in the database are a match. When both of them are verified, we use the MySQL query of *LOAD DATA LOCAL INFILE*, which appends the entire CSV record to the database, as long as they are new records. The code does not overwrite any existing records. The function then returns either a success or failure message.

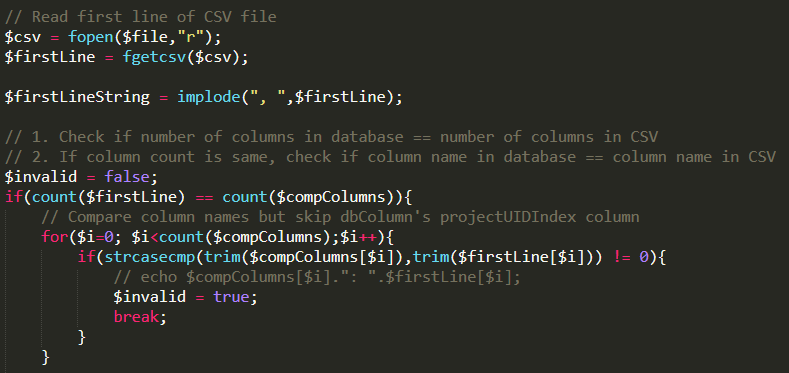


Figure 47: Translating first line of CSV to JSON array

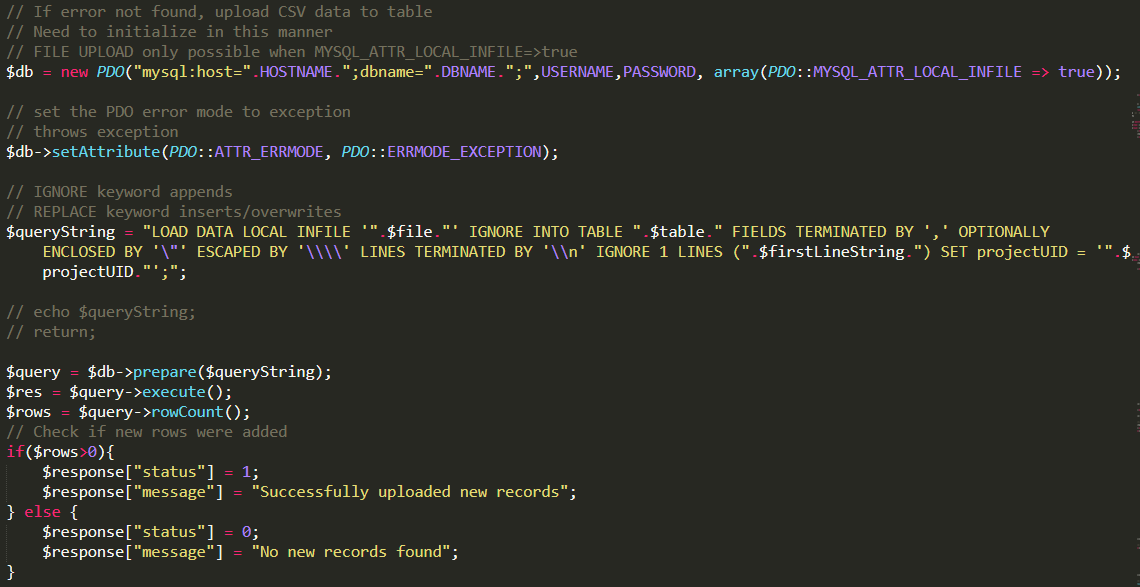


Figure 48: Insert CSV into database using LOAD DATA INFILE syntax

Similarly, administrators and biologists are able to download tabular data from the database depending on the projects assigned to the user. Previously when adding users, projects are assigned to them, which are stored in the database in the *tblproject\_user* table. When downloading the records, that table is considered and the data in any or all table(s) that is/are being downloaded are filtered with the information obtained from the tblproject\_user records. As with uploading the data, the users select the project they would like to download data from initially. At which point the downloadable tables are displayed in a dropdown for them to select and download. The project and table details are sent through jQuery, which obtains the data using SQL queries. The values are returned in JSON array, and using a *JSONtoCSVConverter* function in Javascript, the JSON array is converted to a CSV file and made available for download to the user.

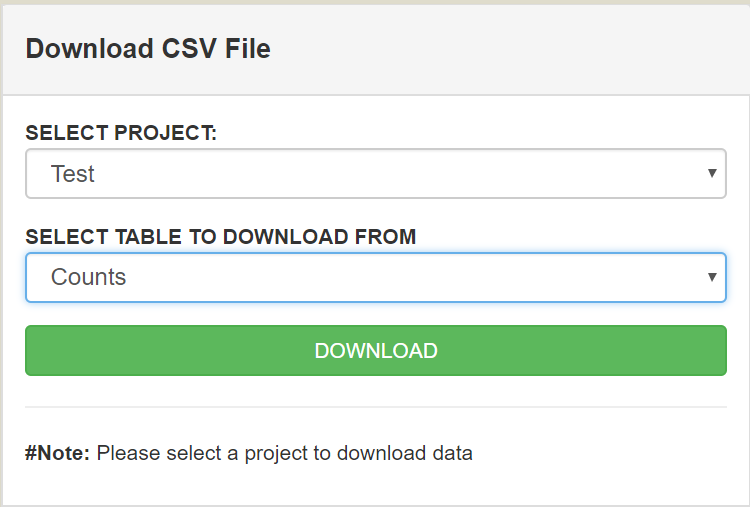


Figure 49: Download CSV Interface

#### AJAX Handlers

There are some instances where data needs to be send from the UI to the backend PHP codes to update existing records in the database or retrieve database records. For instance, to implement the lazy loading in the treeview, requests must be sent every time a node is expanded and each request is done in a similar fashion but the data being sent and retrieved is completely different. Having a single function handle this dynamically makes the process easier, but having the function in the same page as treeview can increase the lines of code significantly and make the code less manageable. Thus ajax-handlers files were created, which would separate the incoming requests dynamically based on the parameters being sent, and proceed to getting the respective information from the database and returning it back to the front-end.

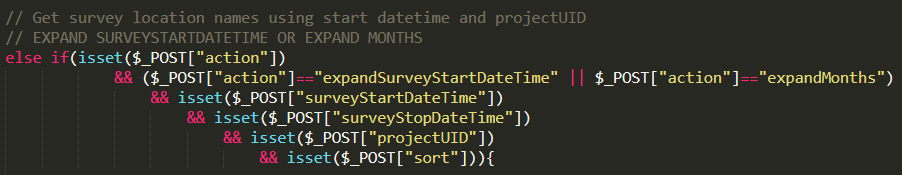
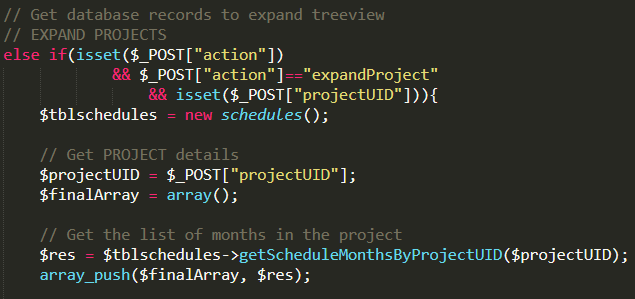
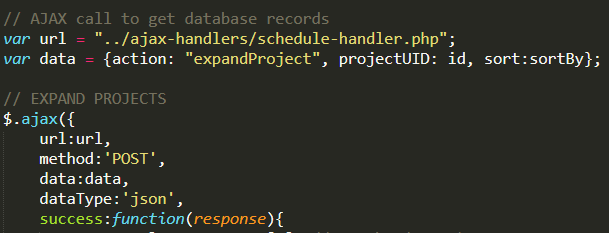


Figure 50: Request for schedule handler and action-handling

#### PHP Data Object (PDO)

There are multiple MySQL APIs available for communication between PHP and MySQL (mysql, mysqli and PDO). The preferred choice due to its object-oriented aspect, the ability to use named parameters and prepared statements to execute queries is the PDO. Utilizing PDO, the coding is done through classes, objects, and methods. For each table in the database, a separate class is created in the “class” directory and implementation of any CRUD operations for that table can be done through functions created within the class.

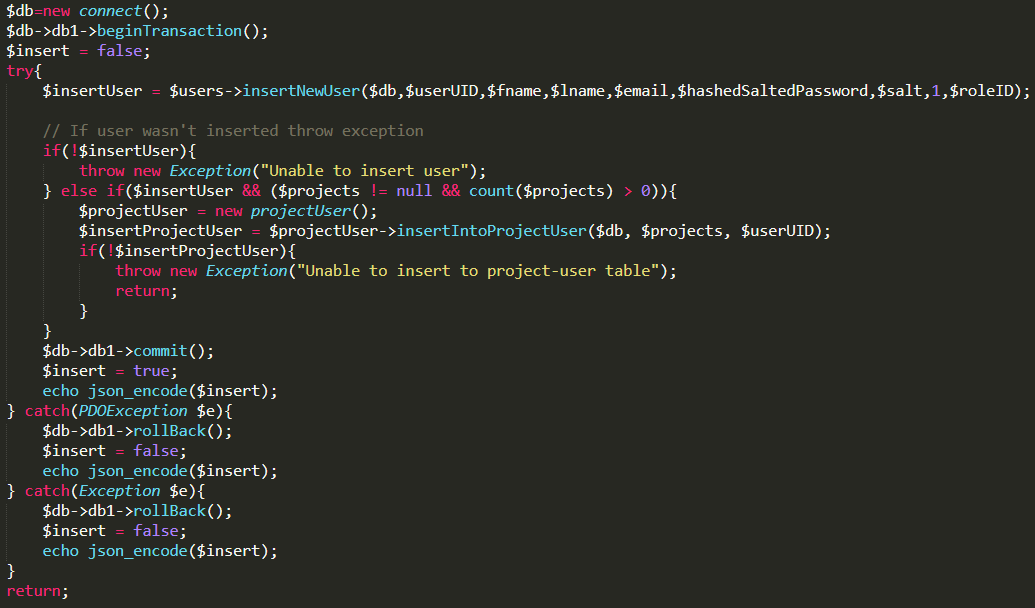


Figure 51: PDO SQL Transactions

In some instances, multiple successive queries need to be executed on the database and failure in any one of them can lead to an error in the database. To solve this problem PDO provides the ability to execute transactional queries on the database. Transactions ensure the ACID (Atomicity-Consistency-Isolation-Durability) properties of a database are preserved. Using transactions allows developers to execute multiple queries on the database and rollback the entire transaction if one or more of the queries returned with an error.

### Notable Features on Creel Survey Web Dashboard

There are some useful functions and tools used in the application, which helped in developing the application and making it more secure, efficient and user-friendly.

1. Every table in the database has a primary key, which is meant to be a unique ID. However, using the default *Autoincrement* method is insecure and obsolete. The unique ID needs to be long and random enough that there is microscopically low chance of any conflict between two IDs in the same table to prevent errors in inserting the records in the database. Thus, to implement this a “Random Number generator” function was used to create a 128-bit Globally Unique Identifier or GUID using hexadecimal characters for every position in a 32-character string, e.g., 30dd879c-ee2f-11db-8314-0800200c9a66 [29].

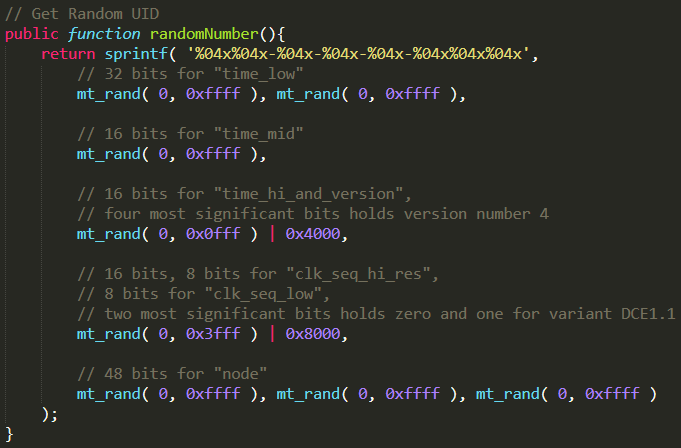


Figure 52: Random Number Generator

1. The images caught for the fish from the Creel Survey Mobile Application are stored in the database Binary Large Objects (BLOBs). When retrieving the value to display on the front-end, the string “data: image/jpg; base64,” needs to be concatenated with the BLOB value. This then converts the image from a Binary object to an image.
2. A useful library used in the dashboard, which was also used in the mobile app was DataTables, which assists in displaying data conveniently, provides sorting and searching capabilities and is capable of providing features such as horizontal scrolling in tables with numerous fields, which was the case for the angler data.

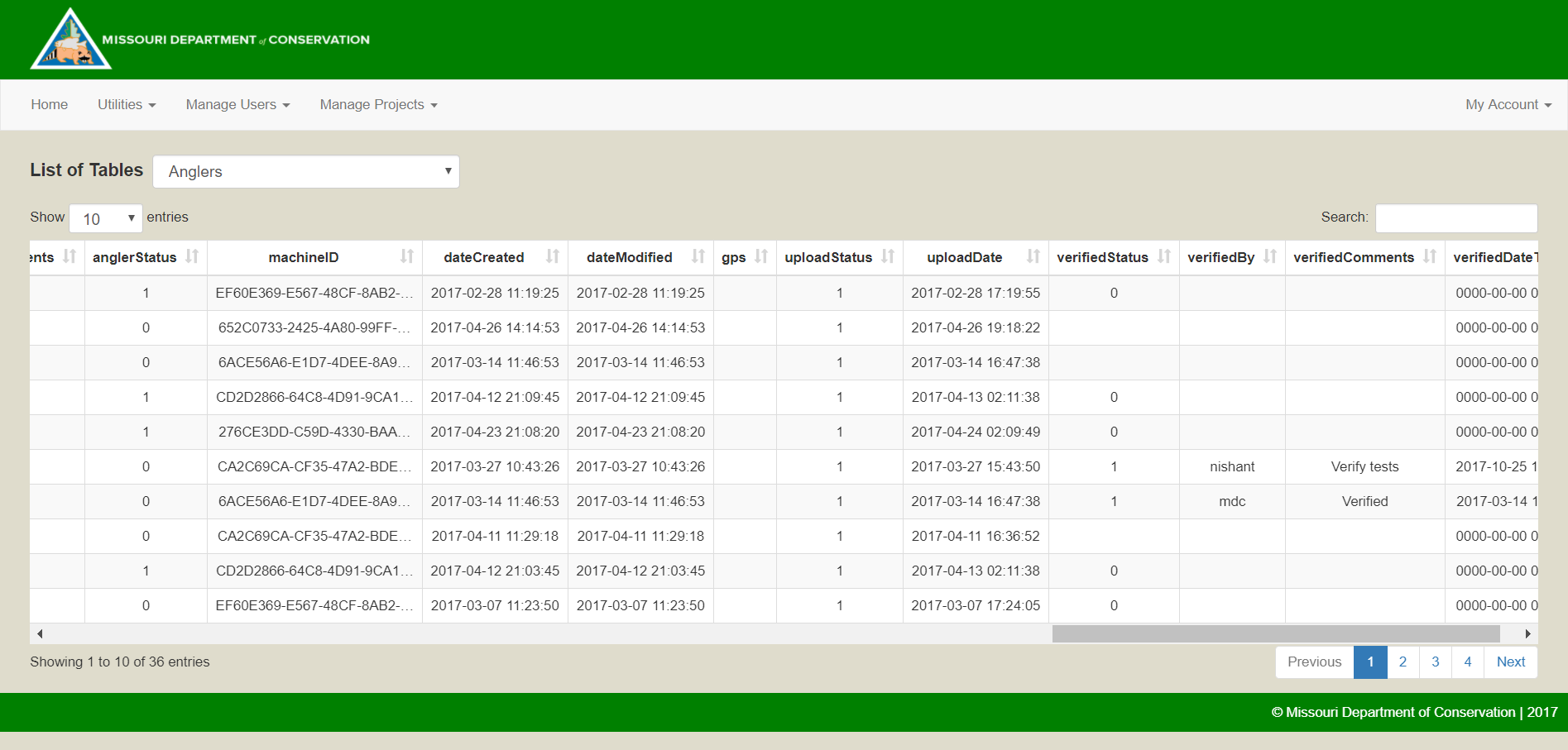


Figure 53: View Raw Angler tables - Datatable implementation

# Results

This chapter discusses the significance and usefulness of the Creel Survey Mobile Application in gathering, storing and organizing the data on the mobile device. It also covers the convenience, usability, and impact of real-time data management environment to the biologists and administrators through the Creel Survey Web Dashboard.

With CSDMA, the MDC staff has a more convenient, usable and automated system making the slow, tedious and bulky paper-based logging process obsolete. With the Creel Survey Mobile Application, clerks are able to enter metrics, take responses and upload data without having to wait for manual data entry process. The application has even extended the original process by being able to take pictures of the fish caught by the anglers. By utilizing the hybrid application, clerks are able to interact with a fluid and responsive interface. The requirements provided by MDC at the start of the application have been met and with the cyclical improvement process, most of the bugs and issues have been resolved making the system even more user-friendly as the development process continues. The web dashboard provides another usable interface for the administrators and biologists verifying the data. The responsiveness and portability of the application means the administrators and biologists can login, view and verify data from anywhere in the world.

A single query takes less time and is less memory-intensive on an application as compared to multiple queries. Keeping this in mind, the pre-requisite tables were uploaded from the Creel Survey Mobile Application as a single JSON array, with most of the work being done at the more computationally capable remote server. Using a database such as SQLite, which allows for SQL queries to do CRUD operations provides a more convenient way of setting and storing data on the mobile device as compared to storing it in a file-based system. Implementing the treeview on the web dashboard provides users with a clear and understandable format for how the data is structured, while lazy-loading improves the overall performance of the treeview itself.

By salting and hashing, user’s password is made more secure before storing it in the database over storing the password directly. It will be extremely difficult and costly for anyone attempting to break the security measures implemented in through the application. Moreover, until the record is reviewed and authorized by an administrator, they will not be permitted to login to the system, which adds an extra layer of security to the application.

One of the major impact of this application is the ability to work in real-time. With the paper-based logging system, MDC staff would, depending on the amount of creel surveys, spend 15-20 hours per month manually entering data into the database for verification and analysis. However, with the integration of the mobile application and the web dashboard, survey data as soon as they are uploaded can be viewed and verified by administrators and biologists. This will help address any inconsistencies in the data earlier than ever before.

|  |  |  |
| --- | --- | --- |
|  | Paper-Based Logging | CSDMA |
| Convenience | **🞭** | **🗸** |
| Reliability | **🞭** | **🗸** |
| Speed | **Slow** | **Fast** |
| Security | **🞭** | **🗸** |
| Automated | **🞭** | **🗸** |
| Real-Time Data | **🞭** | **🗸** |
| Web Accessibility | **🞭** | **🗸** |

Table 2: Difference between paper-based logging and CSDMA

# Conclusion and Future Work

The Creel Survey Data Management Application integrates a mobile and a web-based application to provide conservation staff with a convenient, portable, reliable, and efficient data management solution. The Creel Survey Mobile Application, developed successfully using Phonegap and Cordova, provides the conservation staff with the ability to take data efficiently and in a much more convenient manner than ever before. The verification process is done in real-time, expediting the entire process by hours each month.

## Contribution and Achievement

The development of the CSDMA was a collaboration between MDC and University of Missouri – Columbia’s Computer Science department. My contribution on the mobile application involves large amount of design and development. The Hybrid mobile application provided a newer, and modern insight into mobile development, which I had not explored before and enabled creation of native applications without restricting myself to Android or iOS. Integration of SQLite into the application, implementation of Bootstrap design, adding Fastclick to improve the touch-response times on iOS, data handling of tabular data in the angler view, and development of the server-side scripting to parse the uploaded data were all part of my responsibility. Moreover, the web dashboard was developed from scratch, and with feedback from MDC and other members of the CS lab, I was able to develop the entire interface. The initial design of the database was provided by MDC but with coordinated discussions, we came up with a more refined schema, which better fit the requirements.

## Future Work

As the first implementation of a digital framework to manage data, CSDMA provides a robust framework for any future enhancements. The mobile application, if expanded to Android devices will only require minimal effort due to the compatibility provided by Hybrid application development. Even though the mobile application works efficiently, the performance of the application can be further improved by potentially implementing it as a Single Page Application (SPA) over the current Multi-Page Application. SPAs have become popular in the past year with the implementation of Angular in the Ionic framework. Similarly, adding visualizations and statistical measurement tools to the web dashboard can create a more detailed application capable of producing data analytics to MDC.

# References

|  |  |
| --- | --- |
| [1] | A. Smith, "Record shares of Americans now own smartphones, have home broadband," January 2017. [Online]. Available: http://www.pewresearch.org/fact-tank/2017/01/12/evolution-of-technology/. [Accessed 11 September 2017]. |
| [2] | J. Schobel, R. Pryss, M. Schickler, M. Ruf-Leuschner, T. Elbert and M. Reichert, "End-User Programming of Mobile Services: Empowering Domain Experts to Implement Mobile Data Collection Applications," in *IEEE International Conference on Mobile Services*, San Francisco, 2016. |
| [3] | M. Trucano, "Using mobile phones in data collection: Opportunities, issues and challenges," World Bank, 18 April 2014. [Online]. Available: http://blogs.worldbank.org/edutech/using-mobile-phones-data-collection-opportunities-issues-and-challenges. [Accessed 11 September 2017]. |
| [4] | N. Bamwine, "Mobile Phone Based Data Collection for M&E and Evidence Based Reporting," LinkedIn, 17 August 2016. [Online]. Available: https://www.linkedin.com/pulse/mobile-phone-based-data-collectionfor-me-evidence-nelson-bamwine/. [Accessed 02 October 2017]. |
| [5] | Z. Peng, N. Wergeles and Y. Shang, "Development of The Missouri Department of Conservation Application: A Mobile Data Collection Platform For Natural Resource Management," Computer Science Department, University of Missouri, Columbia, 2015. |
| [6] | S. Sheriff, M. Tomlin-McCrary and S. Gao, "Creel Survey Workshop," Missouri Department of Conservation, Columbia, 2013. |
| [7] | J. Cotton, "Data: The importance of accuracy, integrity and real-time integration," Information Bloggers, 15 December 2014. [Online]. Available: http://www.informationbuilders.com/blog/james-cotton/19152. [Accessed 7 October 2017]. |
| [8] | Telmediq, "The Importance of Real-Time Data Entry to the EHR," Telmediq, 20 April 2017. [Online]. Available: https://www.telmediq.com/news/importance-real-time-data-entry-ehr. [Accessed 2017 7 October]. |
| [9] | N. M. A. Munassar and A. Govardhan, "A Comparison Between Five Models Of Software Engineering," in *IJCSI International Journal of Computer Science Issues*, 2010. |
| [10] | A. Powell-Morse, "Iterative Model: What Is It And When Should You Use It?," Airbrake.io, 15 December 2016. [Online]. Available: https://airbrake.io/blog/sdlc/iterative-model. [Accessed 7 October 2017]. |
| [11] | Tutorials Point Pvt. Ltd., "SDLC - Iterative Model," Tutorials Point Pvt. Ltd., 2017. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc\_iterative\_model.htm. [Accessed 7 October 2017]. |
| [12] | Apache Cordova, "Documentation - Apache Cordova," Apache Cordova, 2017. [Online]. Available: https://cordova.apache.org/docs/en/latest/. [Accessed 7 October 2017]. |
| [13] | T. Hornby, "Salted Password Hashing - Doing it Right," Code Project, 14 February 2016. [Online]. Available: https://www.codeproject.com/Articles/704865/Salted-Password-Hashing-Doing-it-Right. [Accessed 8 October 2017]. |
| [14] | R. J. Camp and J. M. Wheaton, "Streamlining Field Data Collection with Mobile Apps," *Eos, Transactions, American Geophysical Union,* vol. 95, no. 49, pp. 453-472, 2014. |
| [15] | B. Ueberschär, M. Bourimi, D. Kesdogan, C. Neumann, O. Dreisbach, J. Dachs and W. Schwittek, "Mobile Applications for the Angling Community," in *6th World Recreational Fishing Conference*, Berlin, 2011. |
| [16] | J. T. Papenfuss, N. Phelps, D. Fulton and P. A. Venturelli, "Smartphones Reveal Angler Behavior: A Case Study of a Popular Mobile Fishing Application in Alberta, Canada," *American Fisheries Society Journals,* vol. 40:7, no. 7, pp. 318-327, 2015. |
| [17] | W3, "Web Storage (Second Edition)," W3, 19 April 2016. [Online]. Available: https://www.w3.org/TR/webstorage. [Accessed 8 October 2017]. |
| [18] | M. Guay, "The 20 Best Online Survey Builder Tools," Zapier, 24 June 2016. [Online]. Available: https://zapier.com/learn/forms-surveys/best-survey-apps/. [Accessed 9 October 2017]. |
| [19] | C. Cohn, "Build vs. Buy: How to Know When You Should Build Custom Software Over Canned Solutions," Forbes, Inc., 15 September 2014. [Online]. Available: https://www.forbes.com/sites/chuckcohn/2014/09/15/build-vs-buy-how-to-know-when-you-should-build-custom-software-over-canned-solutions/#2416028cc371. [Accessed 9 October 2017]. |
| [20] | SurveyMonkey, "SurveyMonkey Plans and Pricing," SurveyMonkey, 2017. [Online]. Available: https://www.surveymonkey.com/pricing/details/. [Accessed 9 October 2017]. |
| [21] | K. Nath, "The importance of UI and UX in mobile apps," Mobiloitte Blog, 14 December 2016. [Online]. Available: http://www.mobiloitte.com/blog/importance-ui-ux-mobile-apps. [Accessed 12 October 2017]. |
| [22] | R. Bord, "Why Is Wireframing Your Mobile App So Important?," Upwork, [Online]. Available: https://www.upwork.com/hiring/for-clients/importance-of-wireframing-mobile-apps/. [Accessed 12 October 2017]. |
| [23] | Gartner, Inc., "Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016," Gartner, Inc., 15 February 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3609817. [Accessed 15 October 2017]. |
| [24] | J. Stangarone, "The Mobile App Comparison Chart: Hybrid vs. Native vs. Mobile Web," mrc (US), 14 June 2016. [Online]. Available: https://www.mrc-productivity.com/blog/2016/06/the-mobile-app-comparison-chart-hybrid-vs-native-vs-mobile-web/. [Accessed 15 October 2017]. |
| [25] | G2 Crowd, Inc., "Best Mobile Development Frameworks Software," G2 Crowd, Inc., [Online]. Available: https://www.g2crowd.com/categories/mobile-development-frameworks. [Accessed 15 October 2017]. |
| [26] | Adobe Phonegap, "About | A high-level summary of what PhoneGap is all about.," Adobe Phonegap, [Online]. Available: https://phonegap.com/about/. [Accessed 15 October 2017]. |
| [27] | A. Trice, "PhoneGap Explained Visually," Adobe Phonegap, 2 May 2012. [Online]. Available: https://phonegap.com/blog/2012/05/02/phonegap-explained-visually/. [Accessed 16 October 2017]. |
| [28] | J. Lithner, "Fast TreeView," Code Project, 9 November 2013. [Online]. Available: https://www.codeproject.com/Articles/679563/Fast-TreeView. [Accessed 25 October 2017]. |
| [29] | E. Lippert, "Guid guide, part two," 30 April 2012. [Online]. Available: https://ericlippert.com/2012/04/30/guid-guide-part-two/. [Accessed 25 10 2017]. |