11-2014

Client/Server Data Synchronization in iOS Development

Dmitry Tumanov

Client/Server Data Synchronization in iOS Development

A thesis

presented to

the faculty of the Department of Computing

East Tennessee State University

In partial fulfillment

of the requirements for degree

Bachelor of Science in Computer Science

Honors-in-Discipline program

---

by

Dmitry P. Tumanov

November, 2014

College of Business and Technology: Honors-in-Discipline

College of Business and Technology Honors Thesis Signature Approval Form

Client/Server Data Synchronization in iOS Development

November 17, 2014

The members of the Thesis Committee approve the Senior Honors Thesis for *Dmitry Tumanov*	____________________
Thesis Committee Chair *Prof. David Tarnoff*	____________________
CBAT Thesis Committee Member *Dr. Martin L. Barrett*	____________________
External Thesis Committee Member *Mr. Murad Ayoub*	____________________
CSCI HID Coordinator *Dr. Christopher Wallace*	____________________
CBAT Honors Director *Dr. Tom W. Moore*	____________________

ABSTRACT

Electronic gadgets such as touchpads and smartphones are becoming more popular in business and everyday life. The main advantage of mobile devices over personal computers is their portability. Cellular data plans allow Internet access without having permanent access point. There is a number of web-based applications available for gadgets. The primary goal of these apps is to provide their services through constant Internet access. However, it may affect the operation of both devices and applications. The objective of this thesis is to find a better way of client/server data synchronization in iOS development that can reduce the negative consequences of constant Internet access such as Internet traffic consumption, reduced battery life and cover area limitation of cellular services. As the result of the conducted research, the "FollowQTM" application was developed. Despite the fact, that the application is synchronized with web server, it does not require constant Internet access to operate.

DEDICATION

I dedicate this work to my loving parents who supported me during my education.

ACKNOWLEDGEMENTS

I would like to express my appreciation to everyone who supported me during my preparation for this thesis, especially Dr. Wallace, Dr. Barrett and Mr. Ayoub for their time and help. I also want to thank Professor Tarnoff for his significant contribution into this work.

Contents

List of Figures

1 INTRODUCTION

1.1 Background

The following chapter will introduce the objectives of the client/server data synchronization in iOS development research. It will discuss the relevance of the thesis to the problems of applications created for portable devices such as smartphones and touchpads. These problems include high data consumption and hardware limitations that can affect the device usage. Then, it will discuss how the conducted research may reduce these problems through the software implementation.

1.2 Relevance of the Research

The high-tech market saw the growth of portable devices such as smartphones and touchpads. It is influenced by their size and usability. Despite the fact that they are pocket sized, modern gadgets do not lose by performance to some modern personal computers. They contain multi-core processors and high memory capacity. Besides advanced hardware, gadgets come with progressive operating systems such as iOS, Android and Windows Phone. Applications created for these platforms became as popular as software products for personal computers. Literally, gadgets replaced PCs for certain people. The wireless connection lets users stay online even away from stationary access points. This fact opened new horizons for the software development and for business in general.

Even though it is undeniable that smartphones and tablets now are close in performance to modern computers by means of hardware capacity and performance, there are constraints that should be taken into concern by the developers. One of these limitations is battery life. Devices constantly consume electric energy. As more resources are engaged in the operation of an application, the more energy is absorbed and consequently the less time is left for a device operation until it is charged. Additionally, the wireless services such as AT&T, Verizon and T-Mobile do not cover the entire country with cellular access points.

Therefore, there are chances that web based apps would be unusable in uncovered areas. The third constraint also applies for Internet services. Unfortunately, the data plans that providers offer to their consumers are unreasonably expansive. Thus, many users limit their data usage.

1.3 Objectives

The three constraints discussed are especially relevant for Internet based applications. The following research will try to find the solution to the problem of client/server data synchronization in iOS development in order to limit battery consumption, data usage and eliminate operational dependencies on data access points. The "FollowQTM" app will be developed along with this research to support received data. The goal of the app is to mimic the "Follow the Quilt Trail" website. The data of the application should be synchronized with the website with the least amount of dependencies on wireless connection.

Research will start with Review of Literature section that will discuss the work done with library materials. It will contain the information gathered by a number of scientists and developers in similar fields of study. It will discuss the importance of mobile applications in tourism, followed by the influence of the mobile applications on the smartphones usage. Then it will introduce the data management options in iOS development followed by the discussion of three different serialization techniques and their advantages and disadvantages. Finally, this chapter will compare threading options in iOS.

The Methods and Procedure chapter will discuss steps that will be accomplished to develop the "FollowQTM" app. It will start with gathering the requirements for the product followed by the design of the user interface. It will then cover the two milestones of the implementation phase. The first milestone will include the development of the app's core functionality. The second milestone will cover the development of the data update functionality. The last part of the chapter will cover the testing that will be provided after implementation phase.

The final chapter of the thesis is Results and Conclusion. The goal of this chapter is to wrap up the results of the thesis. It will discuss the results of the developed iOS application with client/server data synchronization. It will also cover the open issues that were met during the development process. Finally, it will provide the conclusion of the research.

2 REVIEW OF LITRATURE

2.1 Background

In order to perform the study discussed above, several topics were reviewed during the preparation for research. The following section is directed to discuss the work that has been previously accomplished in the fields of studies that will become the foundation of this research. It started with a discussion of the importance of mobile applications for modern business applications followed by the review of currently available local data storage interfaces that can be used in the development of the “FollowQTM” application. Also, it is worth discussing the available serialization techniques that could be used during client/server data exchange over TCP/IP network.

2.2 Value of Mobile Applications in Tourism

It is hard to imagine the growth of modern business without using high technologies in their operation. Smartphones and tablets along with the services that they provide are a great source of optimizing business processes, communication and even advertisement for existing or potential customers. The popularity of gadgets constantly spreads due to their usability and availability. It is possible to control the whole factory execution through the portable devices beginning with loading raw materials to a line and finishing by packing an end production. Also, gadgets have improved the communication between people with VoIP applications. It may save much money that could be spent paying for roaming.

Particularly, the mobile services that are built in smartphones and tablets may improve one’s traveling experience. There is a number of apps that were specifically built for tourists all over the world. According to Dewey, “When it comes to mobile phone use for travel specifically, 53 percent of travelers are utilizing apps and 47 percent are utilizing their mobile web browser. That follows the overall trend of smartphone users - with 83 percent using apps and only 17 percent using their mobile web browser” (Dewey, 2013). So, most tourists are using mobile apps during their vacations in order to improve the quality of recreation. Since the information and services they are looking for, including flight booking, hotel booking and attractions, are available through such apps. Also, GPS technologies are helping tourist to find directions if they are travelling by car or walking. All of the above make the trips more productive and convenient, decreasing the amount of time spent for trip preparation.

Several years ago there was no source available for tourists other than their friends and people they know to get information about the quality of hotels, bars, restaurants, etc. However, now they can access reviews of other peoples' experience via tourism mobile apps. Dewey states that, "Fourteen percent of travelers reported learning about travel deals or events through social networking, an increase of 7 percent since 2011. Those posting their own travel reviews grew from 12 percent in 2011 to 25 percent in 2012" (Dewey, 2013). According to these numbers one can make the conclusion that people increased their trust in such reviews and became more active in submitting their own reviews in such apps.

2.3 Influence of Internet Based Apps on the Smartphones Usage

Despite the fact that traveling became easier with the development of smartphones, most of these applications are constantly utilizing an Internet connection in order to synchronize with server data and applications, thus they became limited once an Internet connection was lost. Other problems with maintaining an Internet connection include limited usage areas, and high battery consumption

First of all, Web-based applications require an Internet access and traffic usage. With everyday usage of such apps, cellular service can become costly. For example, AT&T charges their clients $20 for 300mb of traffic and $30 for 3Gb (AT&T, 2014). If a user exceeds data limits, he/she would pay twice as much. Most people do not want to overpay for data plans. Therefore, many people prefer to choose the less expensive data plan and keep the cellular connection idle most of the time. Instead, they would use online services with Wi-Fi connections that are only available in certain areas

Secondly, despite the fact that modern providers like AT&T, Verizon, and T-Mobile have great coverage areas, there are still rural locations where no Internet access may be found, but services are still needed. For example, there are several traveling applications including the application that will be created along with this thesis, which provide real-time information about rural zones that one would like to visit. It is likely, however, that tourists will not find wireless connections in such places. Thus, it may become impossible to access Web-based application data; and as a consequence, the application may become useless.

Thirdly, the portable nature of a mobile device means power consumption may be an issue without an electric source. According to Ding and Wing, "despite the tremendous market penetration of smartphones, their utility has been and will remain severely limited by their battery life" (Ding et al., 2013). Continuous Internet access uses a significant amount of battery power, which can reduce battery life and consequently limit the usage time. As a result, many users would choose to turn their wireless connections off in order to save their battery's life and keep essential phone functions such as calling and instant messaging services.

2.4 Data Management in iOS Apps

Since "Follow the Quilt Trail Mobile" is a data driven application, an efficient and simple data storage format shall be chosen. In accordance with the Apple Developer website, there are a number of options available for developers to meet their data management needs. These options include Core Data, SQLite, XML files and network connections. Each option can be usable in the development of data driven applications.

According to Apple's website "Core Data provides a flexible and powerful data model framework for building well-factored Cocoa applications based on the Model-View-Controller (MVC) pattern" (Apple, 2014a). Core data provides a convenient method for data management in iOS applications using graphical tools that are available in the Xcode integrated development environment. It lets the developer organize his/her objects in a way that can be used for both small and big projects.

In order to manipulate data that is stored in persistent storage, for instance a file, a developer needs tools to read, write or delete. If one wants to support undo functionality, an application must track changes as well thereby adding another complexity level for a developer. However, Apple offers Core Data allowing developers to effectively manipulate application objects. “Using the Core Data framework, most of this functionality is provided for you automatically, primarily through an object known as a managed object context (or just “context”)” (Apple, 2014a). Managed Object Context allows access to the persistent store coordinator which acts as an intermediary between application objects and data stores (Apple, 2014a). Besides the ability of manipulating objects, the context lets an application keep track of changes that adds the ability to undo changes if needed.

It is also possible to retrieve data through the managed object context in a similar way as working with database management systems. In Core Data this process is called fetching request. Fetching request objects consist of three parts to specify the data that an application needs to access. The first part must contain the name of an entity, the second part should contain the predicate to the data, and the third part specifies the order in which one wants to retrieve objects.

Extensible Markup Language (XML) is a convenient way to organize, structure and store data in a way that is both human readable and easy to manipulate from the application side. Even though this format is rather new, it has already shown its advantages and now is in the toolbox of a number of software developers. The XML format might look similar to HTML (Hypertext Markup Language), however the tags used are created by the developer. The final goal of HTML is to display data, XML on the other hand doesn’t do anything to data, but it stores it in a convenient way using tags.

XML is a cross platform language and can be integrated into any application or device. According to Fairbanks, “XML’s platform independence and extensibility allow it to easily read and generate structured documents in any environment. Thus, XML has become the language of choice for managing, creating and distributing data in all digital media" (Fairbanks, Gribbons, Nybo, Pean, & Wright, 2002). Since there is a plan for future development of "Follow the Quilt Trail Mobile" for both iOS and Android platforms, XML would be acceptable tool to utilize.

Another option for an iOS developer who wants to create data driven applications is the SQLite library. SQLite is lightweight and easy to integrate database engine powered by framework that is part of Xcode installation. According to Lin and Neamtiu, "SQLite a lightweight, zero-config, server-less SQL engine encapsulated into a library that can be simply linked into an existing application and used via the SQL API" (Lin & Neamtiu, 2009). Therefore, it might be convenient to choose it in order to simplify cross platform development. Since it might be used in both iOS and Android applications, it possible to have one file with stored database and transfer it between the two platforms. Therefore, SQLite will be the convenient solution for data storage in this project.

2.5 Data Serialization Formats

Most data driven applications require synchronization with a server to get updates. On the one end of the communication link is the client application that asks the server to provide it with updated data. On the other end, the server receives the client's request and prepares the required data for transferring. There must be a convenient data structure that can collect data, compress it and send it to the client application. The client application receives this data and translates it to a format so that it can use it. This process requires serialization and deserialization. There are currently several broadly used serialization formats such as JSON, XML, Apache Thrift and ProtoBuf. All of them have both advantages and disadvantages. As discussed previously, "Follow the Quilt Trail" is a data driven application, which should be synchronized with the server to get periodical updates since the database of participating farms increases every month. Thus it will require data exchange with the server and consequently a serialization format to be used.

XML has being used for data serialization for more than 10 years in the operation of web services. The most beneficial feature of XML serialization is that it is text-based and thus, human readable and easily implemented. However, it was developed before the first smartphone was introduced to the market, so it has been criticized for not being "well suitable in mobile environment" (Sumaray & Makki, 2012).

JSON is another text-based serialization format. It was introduced by Douglas Crockford as, "part of a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999" (JSON, 2014). JSON is lightweight and has the main advantage over XML that is lack of markup overhead, that makes JSON objects lighter and consequently cheaper to transfer compared to XML serialization.

There are also relatively new data serialization formats that in contrast to text-based serialization formats, use binary in order to serialize data. There are currently two leaders among binary formats: Apache Thrift and ProtoBuf (Protocol Buffer). Both of them are extremely fast in serialization and deserialization and they are lightweight.

Apache Thrift was initially developed and used by Facebook. Because they are not text based, they do not have the same drawbacks during serialization. Instead of passing a string character by character, they use positional binding letting the name part of the name-value pair to be "stored in a separate file (.proto for ProtoBuf and .thrift for Apache Thrift)" (Sumaray & Makki, 2012). Since these files are binary and subsequently smaller than a text file representing the same data, they can greatly decrease the amount of network traffic. However, these files need to be compiled, so both Apache Thrift and ProtoBuf are not cross platform, they were built to be supported by a limited number of programming languages. "Nowadays, Apache Thrift supports C++, C#, Erlang, Haskell, Java, Objective C/Cocoa, Perl, PHP, Python, Ruby, and Squeak", while ProtoBuf supports C++, JAVA, and Python (Sumaray & Makki, 2012).

One of the most important factors affecting the development of the product is energy consumption and battery life. The main reason to build an energy-safe application is that, once the battery dies, all other essential smartphone functions such as the phone are not able to operate (Oliver & Keshav, 2011). According to Gil and Trezentos' research, "For applications that synchronize data with a Web sever repeatedly power management is a key factor" (Gil & Trezentos, 2011). The data format affects message size and the amount of time needed to send and receive messages, there fore affecting power usage.

In order to reduce the time needed for message transfer and increase the battery life, JSON along with ProtoBuf and Apache Thrift are the best available choices for a developer. Gil and Trezentos state, "Our investigation shows that between XML, JSON and Protocol Buffers, the better format for data synchronization was the JSON format with compression, because has better efficiency in terms of time synchronization, parsing on server and better performance in battery management" (Gil & Trezentos, 2011). In addition to other advantages, JSON is relatively simple to implement serialization on both the client and server sides.

2.6 Threading in iOS Development

"Follow the Quilt Trail's" database is constantly growing since there are more and more farmers who desire to participate all over the North-East Tennessee. Thus, the updates for mobile app are expected to be significant. The amount of time required for applying updates from the server is unpredictable; so it is important keep the main application functionality available for a user during this process and avoid the application stopping while waiting for local data storage to be populated with new data. Threads can help to maintain operation of the application during the process of synchronization and update using the advantage of an iPhone's multi-core processor (starting with the iPhone 4s).

According to Garcia, "Thread is an independent flow of control within a Process. A traditional UNIX process has a single thread that has sole possession of the process' memory and other resources" (Garcia & Fernandez, 2000). Thread is a smaller context and runs within the process sharing global data. However, each thread has its own call stack, local variables and program counter, so it can work with a main thread simultaneously having almost no affect on the main thread's execution (Garcia & Fernandez, 2000).

Nevertheless, the developers, especially the smartphone and embedded system developers, must be concerned about resource limitations. Threads allocate memory in both the program memory and in kernel memory. Thus, inefficient usage of threads can affect the performance not only of the application, but a system in general. Apple's iOS provides developers with a variety of threads, though the most popular and easy to use are NSThread interface and POSIX threading library. Both of these are relatively easy to implement and integrate in an app.

Since NSThread is the part of Cocoa library, it is easy to integrate in Objective C source code without worrying about losing the self-descriptive elegance of this programming language. Moreover it is relatively simple to handle with class methods that can meet all of the developer's needs. In order to create a new thread, the application should either call `+(void)detachNewThreadSelector:(SEL)aSelector toTarget:(id)aTarget withObject:(id)anArgument` or `+(void)start`.
A developer should be careful with the first method of starting a new thread since if the application is not set up for automatic garbage-collection, he/she should set up autorelease for the detached thread in the method identified by aSelector and release it once the thread has completed its tasks (Apple, 2014b).

Objective-C along with C++ is derived from the C language. This means a developer is able to add C code in his/her application and be sure that it will run on any of Apple's platforms. This provides the developer with ability of cross platform programming. Since multiple APIs already exist that were written using the C language, iOS developers have access to a number of advantageous libraries that have been developed over several decades. This includes the POSIX thread libraries.

Portable Operating System Interface (POSIX) thread API that was initially created in the 1980s in C is also available for iOS application developer since it was developed for UNIX (iOS is a UNIX like operation system). The IEEE Technical Committee on Operating Systems standardizes thread specifications such as POSIX, the standard for which is POSIX 1003.1 (Garcia & Fernandez, 2000).

The POSIX library is now part of Xcode, so no additional libraries need to be installed in a development work station. In order to access the POSIX library, a developer should include the pthread.h file. The following command will start a new thread:

```
int threadError = pthread_create(&posixThreadID, &attr,
&PosixThreadMainRoutine, NULL).
```

`PosixThreadMainRoutine` is a method that will be running as a separate thread. In order to kill the thread, one should call `returnVal = pthread_attr_destroy(&attr)` (Apple, 2014b).

3 METHODS AND PROCEDURE

3.1 Background

The proposed “Follow the Quilt Trail” mobile application will provide users with most of the functionalities of the “Follow the Quilt Trail” website including advertising and location services. It will be created for Apple products based on customer’s preference. According to her research, more potential users of the app use the iPhone and the iPad. However, in the future there will be a plan to support Android platforms as well.

There are a number of risks that can affect the development process and the quality of the application. The highest of them is a time limit of eight weeks for the application to be developed from scratch and published in Apple’s App Store that was provided by the customer. A lack of experience working with both smartphones and Objective C programming language could also cause a significant delay.

The methods that will be used in order to complete this research and develop an application that will meet all customer needs and Apple’s App Store requirements will be discussed in this chapter. The project flow is decided to be a simplified unified software lifecycle model due to time constraints. The development should go through the process of gathering requirements followed by the design phase. The next step will be implementation of the core functionality followed by implementation of the update functionality. Finally, the application will be tested and deployed.

3.2 Requirements Elicitation

The development of any application requires certain procedures that will become a foundation for the end product. Thus, the development process starts with the requirements elicitation, which is the most important phase. In the article, “Modern Trends Toward Requirement Elicitation,” by Asma Sajid, Ayesha Nayyar and Athar Mohsin, they write, “Accurately capturing system requirements is the major factor in the failure of most of software projects" (Sajid, Nayyar, & Mohsin, 2010). Despite the fact that this idea is well known among developers, the market is full of software that fails due to the mistakes made during the requirement elicitation phase. Therefore, in spite of the time constraint, there is the need for a qualitative list of requirements that will be accepted by both customer and developer

Due to the time limitation for the "Follow the Quilt Trail" development, close collaboration between the customer and the developer is vital. It is possible to miss the required information, or create an unneeded feature due to the lack of communication, especially if the product is to be completed in a short period of time. Thus, it was decided to meet weekly with the customer and discuss the progress of the development and features that should be changed or added. The goals of "Follow the Quilt Trail Mobile" can only be achieved through high cooperation. Usually the requirement elicitation should take a significant amount of time and effort since it is the foundation of the future software product. Developers should prepare documentation such as the Software Requirement Specification, create user stories, use cases and risk management plan before moving to the next design phase. However, when there is not sufficient amount of time to prepare complete documentation, developers should reduce the list to contain only those documents that will be the most crucial for both quality and time. Therefore, it was decided to meet three times during the first week with the customer in order to create the artifacts required for the development process that include prcis, the list of requirements, top-level use case diagram and the use cases.

During the first week of the project the following list of requirements was created:

- The system shall be easy to use.
- The system shall support all versions of iPhone.
- The system shall support current iOS version (iOS V6.1 by the time of the discussion).
- The system shall provide user with location services and directions.
- The system shall store data in the local data store.

- The system shall provide the following barn data: address, phone number, barn picture, and barn description.
- The system shall be synchronized with the server.
- The system shall display sponsors in the main menu.
- The system shall provide users with ability to donate to the project via customer's web site.
- The system shall be supported by iPhone 4, iPhone 4s, iPhone 5, iPhone 5s, and iPhone 5c.

These are the main requirements that were approved by both the developer and the customer. They will be used in the next steps of the development process.

3.3 Design

The next phase in the development of "Follow the Quilt Trail Mobile" is the design. According to the requirements gathered, the application should be easy to use since the target users ages may vary. Thus, a GUI that includes menu, navigation, information and maps should be both intuitive and attractive. In general, all of the menu options will provide the user with a way to search for a particular barn or learn about barns in their area. There will be eight views to fulfill all requirements.

1. Main menu
2. County List View to Barns List View
3. County List View to Map View
4. Favorites List View
5. Barns List View

6. Map View

7. Detail View

8. Map Barn View

The main menu is the most important part of the application since it is the first component viewed by users once the app has been downloaded. It should reflect farm tourism as the main idea of the app and be attractive. Therefore, the background will be the picture of one of the barns that participates in the "Follow the Quilt Trail" project. Its quilt was chosen to become a part of the logo that will be placed in upper part of the screen. The logo picture will also be used as the button to get to the screen with the information about the "Follow the Quilt Trail" project. This will also provide a place where the user can donate if they desire.

Besides the logo, the main menu will contain four buttons that will provide a user with four options. The first option will forward the user to the County List View with a list of available counties. From this view, the user will be forwarded to the list of barns available for the selected county. The Barns List View will represent a table view that will list barns that participate in the "Follow the Quilt Trail" project. Each cell of the table will contain the icon representing the quilt for that particular barn. The barn's name will be shown to the right of its icon. Also, the address of the barn is shown below its name to provide the user with the rapid access to the address without loading the detail view. It will make it easier to drive to the barn.The second option will navigate users to the Barns List View directly. However, in contrast to the first option, users will get the list of all barns grouped by counties. The counties' names will be used as headers of the created groups and will be ordered alphabetically.

The third option will also navigate the user to the County List View, but in contrast to the first option, the user will be forwarded to the Map View. The map view will display the map centered in the chosen county. It will show pins that represent the barns and their positions such that the user will be able to choose a barn by its geographical position. Each pin will present an annotation view displaying the icon of the barn, its name and its address.

The fourth option will navigate users to the Favorites View. This view will display the list of barns that the user decided to save for future use. It will contain an edit button that will provide him/her with the ability to delete certain barns. It will also display the amount of barns that are currently in the list.

All four options will ultimately navigate the user to the Detail View. The Detail View will display the bigger picture of a barn in the upper part of the screen. It will also display the barn's name and its address. The address will be stored in the static cell. The user will be able to select it in order to access directions to the barn. Another static cell will contain the phone number of the farm if it is stored in the database. If the viewing barn was saved to favorites, the user will see the blue star next to the barn's name. Otherwise, the static cell will display, "Add to Favorites." If the user selects this option, the barn will be saved to the list of favorite barns. A small description of the farm will be placed in the bottom part of the view. Also, many farmers sell their goods locally, so the application will indicate if the barn's owner has goods for sale.

The most challenging part in the design process is to support different models of iPhones starting with iPhone 3gs. Because of this, all parts of the design should be created for both retina and non-retina displays. The Retina display may contain twice as many pixels in the same surface area as the standard display. Thus the background picture for the starting screen must be created in the following sizes: 320×480 pixels (iPhone 3gs), 640×960 pixels (iPhone 4/4s with retina display) and 640×1136 pixels (iPhone 5, 5s and 5c). In addition to the starting screen, all icons and pictures need to be created in two sizes: one for retina displays and one for no-retina displays. Photoshop will be used in order to create most of the design components of the app.

3.4 Implementation of the Core Functionality

The next phase is the implementation of the core system functionality. It will include the development of most of the client side application other than update functionality. It will cover the creation of all views required for the “Follow the Quilt Trail Mobile” application, connecting them in a meaningful way, creating the initial database and writing the Objective C code that will accomplish functional requirements. Xcode will be the main development environment, however several additional tools such as Notepad++ and bash scripting may be required for smaller parts of the development.

The first step will be the creation of the views required for the application. The Xcode IDE includes a convenient tool called Interface Builder that helps developers create storyboards with views and connections between them in an interactive way. A number of templates are available for iOS applications including Master-Detail Application, Tabbed Application, Page-Based Application, Single View Application, Utility Application and Empty Application. The Master-Detail Application template will be used as it is the most suitable template for the purpose of this app. The Navigator Controller that is created automatically with this template will help control the views and navigation between them in the most usable way without requiring a great deal of additional code.

The next step is the development of the initial SQLite database that will contain records about the barns. The records required for the application will be gotten from http://www.arcd.org/quilttrail/, the official web site of the “Follow the Quilt Trail” project. The database for this site is written using the MySQL engine. The script for creating the database can be easily imported using PHP My Admin service in different formats including the SQL format. Since both MySQL and SQLite databases use the SQL standard, the translation of one to another is simple and does not require significant amount of time and effort even for large databases.

There are several ways in which an SQLite database can be built. The first way is to create a database in a Shell with the sqlite3 [path to the database file] command. If the path was not provided, the temporary database file will be created in the current directory. Once the work is finished, the temporary file will be deleted. After running this command, the user starts the Sqlite3 program and now is able to work with database. The following example creates the table “table1” in the Sqlite3 utility:

```
sqlite>CREATE TABLE table1 (
...>clmn1 int primary key,
...>clmn2 text,
...>clmn3 varchar(33)
...>);
```

The other simpler way to work with SQLite database is to install the Firefox add-on SQLite Manager. Besides the ability to write queries in the regular way, the Firefox’s add-on provides users with the graphical user interface to navigate through, create and delete records. The user also may specify the settings of the database and create indexes. One can choose which tool to use in order to build and manage the database, however, the best way is to combine both tools in order to complete the process faster and more convenient. The initial database schema can be created using Sqlite3 if one has generate SQL script by copying and pasting to the terminal window. Once the database is created, the user may switch to the SQLite Manager to make sure that the records were created in the way he/she expected, and make changes as needed. After the database is generated, the file with the database can be moved to the project directory to make it visible in XCode.

The next step of the development is to add additional frameworks that will be needed during the later development process. In order to be able to manipulate SQLite in an iOS project, the libsqlite3.dylib framework needs to be added. This framework is written in C, so all the function calls must be in C format. Also, as the “Follow the Quilt Trail Mobile” will be using global positioning service, the MapKit.framework also needs to be added to the project. It provides all the tools needed for the positioning functionality of the project including maps, pins and directions. The other framework that will be used is

AddressBook.framework. It will aid in managing the contact information for the barns, such as providing addressing information to the Map View in order to get directions.

The final phase of the implementation of the product is creating classes and writing the code to provide the functionality of the project. There will be four custom classes acting as the objects created from the SQLite database file that will be used throughout the execution of the application. Barn class will be a placeholder for the barns objects; BarnList will be the class responsible for the collection of barns; County class will collect information for a certain county; and finally, CountyList will hold the collection of counties.

All of the classes discussed above will be used as templates for the objects that will be created from the SQLite database. As the application starts, the CountyList and BarnList will be instantiated and populated in the Application Delegate (AppDelegate) class. Application delegate class is responsible for handling application level functionalities such as handling events in the application startup time and application shutdown. Because CountyList and BarnList objects will be used throughout the execution of the application by assisting to organize the data in the tables and map views, it was decided to instantiate them in the

```
-(BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:  (NSDictionary )launchOptions
```

method of AppDeligate class. Even though many developers may deem the usage of global variables as a bad practice, it will allow generating data once the loading of the app is completed and allow using it throughout the execution. Thus, it will save time on reading data from database and storing it to memory any time the user switches between views. Barn class will hold all the information that is needed for the user to get the information about a particular barn including its address, phone number and description. The rest of the classes are created automatically in Xcode's interface builder. Thus, it is outside of the scope of this project.

3.5 Implementation of the Update Functionality

The final implementation part will require the development and integration of both the iOS side and the server side portions used in the operation of the "Follow the Quilt trail mobile". The server will contain the PHP script that will be responsible for accessing the MySQL database, reading data and creating a JSON object such that it will contain updated data readable for the iPhone application. On the other side, the iOS application will be responsible for providing its current database state, receiving the update in the form of a JSON object, processing it and applying changes to the local database.

The first step will include creation of the table "update_log" in the MySQL database. It will contain three fields. The "id" field will contain an integer value that will be incremented automatically once the new record becomes the subject to update. It will be compared against the update state of iOS application in order to generate update the JSON object with wanted data. The second field "barn_id" will hold the identification of the barn record that was updated. The last field is "action". This field will be a placeholder for the action that caused the update. The accepted values for the "action" field are "create", "update" and "delete". If the new record of the "barn" table was inserted, the action field will contain the varchar value "create". If the record was updated, the action field will be set to "update". Finally, if the record was deleted, the "action" field will contain "delete".

The second step is to create the PHP script called "synchronization.php" that will handle server side functionality. Once it receives the "Follow the Quilt Trail" application's request with the current database status of the client, it will connect to the MySQL database and run the query that will output all records that were changed since the client's status was last updated. If there is a new data, the script then will generate a JSON object to reflect all records that have been changed.

The client side is more complicated than server side. It will engage the creation of the updater class, a thread that will run it without intervention from the main application flow. NSlocks will protect shared data from the race conditions.

The updater class shall be able to connect to the SQLite database in order to receive the current update state. The update state will then be sent to the "synchronize.php" script in order to get the JSON object with the data that should be changed. In the case where the update state is current and there are no records available for the update, the connection to the database will be closed and the application will continue its normal operation without any changes. However, if there are updates in the server's response, the Apple Delegate class will ask the user if he/she would like to apply these updates. If he/she decides to apply updates, the updater class will get the JSON object generated by the "synchronize.php" script, then store the data in NSDictionary object in order to be processed by the program. Once the dictionary is created, the data will be passed to one of three methods that will mimic the actions created in the MySQL database. All actions will use the class's SQLite connection in order to run a query corresponding to the action. The method `-(void) handleJasonObject` will be responsible for handling actions and calling methods that are responsible for the action. If a record fails to update, the property errorCounter will be incremented.

Since it was decided to keep the update routine away from the main program flow, the updater's class functionality will run as a separate thread. The thread will be created using NSThread class in the Application Delegate class. It was decide that the best time for the app to check for updates was only when the application becomes active using the class's method `-(void)applicationDidBecomeActive:(UIApplication *)application`. There are two reasons that affected this decision. The first reason is power consumption. If the thread stays active during the entire application execution time, it will consume significant amount of energy and generate significant internet traffic in the infinite update checking loop. The second reason is the inconvenience of unexpected notifications, since there must be a user response regarding whether or not he/she would like to apply available updates. There might also be the case when the user is using direction functionality or reading a barn description and wishes not to be disturbed.

3.6 Testing

Since the size of the application is small, it doesn't require a significant amount of tests. Most of the functionality can be tested directly by using the app. The first part of the implementation will be examined strictly on the user experience. The volunteers who wish to help testing the beta version of the "Follow the Quilt Trail Mobile" will get access to the test binary of the app in their smartphones. After two days of usage they will be asked to write a small review and point out any defects they might find.

The second part of the implementation should be tested in a different way since its functionality requires more data operations. It was decided to follow the method of test-driven development for the client/server synchronization part of the application. The assertion helps the developer test the program during the development process. The function "assert" is broadly used in languages such as Objective C, C and C++. The main goal of this testing is to make sure that certain expected conditions have been met and the execution of a program can be continued. In the case of an assertion fail, the execution halts and displays a warning message so that the code can be debugged. The assertion is useful for unit tests to make sure that the flow of the program produces correct results. Thus, several barn records will be created in a MySQL database that will mimic the new updates available. After that, they will be updated and deleted in a random order. The assertion that will be written inside the iOS app will be responsible for checking if the execution process corresponds to the actions that have been declared on the server side.

4 RESULTS AND CONCLUSION

4.1 Background

The following chapter will describe the results of research. It will include the description of the app that was developed in two versions: “FollowQTM” V1.0 and “FollowQTM” V1.1. Then, it will state the minor open issues that will be solved in the future development followed by the conclusion. The objective of the conclusion is to wrap up the results of the conducted research.

4.2 Results

The “Follow the Quilt Trail Mobile” was developed in two versions: V1.0 and V1.1. “FollowQTM” V1.0 includes the core functionality of the application that was introduced in the previous chapter. It allows users to get access to all functions of the “Follow the Quilt Trail” website without having access to the Internet through the local SQLite database. It includes simple navigation between views. It also provides the access for positioning functionalities through the app.

Despite the fact that the application depends on the local data store, it is lightweight since the pictures in the barn Detail View controller are accessed via wireless connection. However, it does not affect the goal of the application. If there is no connection available, the placeholder for description photo will be exchanged with the logo of the application with the assistance of Reachability class. It was developed as an open source framework that checks the current wireless connections. The application supports iPhone 4/4S and 5/5C/5S. It was released in the Apple’s App Store in September 2013 with the support of iOS 6.1. Currently, there are more than 230 downloads. The link to the download is https://itunes.apple.com/us/app/followqtm/id702387611?mt=8.

The “FollowQTM” V1.1 has not yet been released since there are several open issues that will be discussed later. It supports the same hardware as V1.0. However, it also was redesigned in order to be consistent with iOS 7.1. Besides the modern design, it includes update functionality. Now, the customer can change the data of the MySQL server without concerns of the inconsistent data provided by the app and the website. The synchronization was accomplished through the creation of DataUpdater class that sends requests to the server containing its update state and receives the response in the form of JSON object. It asks the user if he/she wants to apply new updates and saves them to the SQLite database. This part was tested and is ready for release after debugging several issues.

4.3 Open Issues

Even though the client/server synchronization part is ready for deployment, there are still several issues that can cause insignificant delay of the product release. The first problem is the issue with icons that are part of the Barn Table View Controller. The preset app already contains icons that correspond to the existing records. It is rather trivial to get the new PNG icons for the new records through HTTP; however all files that are submitted as a part of the binary to iTunes Connect are stored in the application bundle. Because of Apple's security concerns, the files that are stored in a bundle cannot be modified. Therefore, there is no way to delete or replace files that have been added to the app through Xcode. However, there is a possibility to add new files to the document folder with the updates. This issue will be discussed with the customer to make sure that this workaround is acceptable.

The other issue is that the target OS for V1.1 is iOS 7.1. However, iOS 8 has been released as of September 17, 2014. Therefore, the "FollowQTM" will require small design changes to support the new operation system. Also, the app now has to support iPhone 6 and iPhone 6 Plus and their screen resolutions of 1334×750 pixels for iPhone 6 and 1920×1080 pixels for iPhone 6 Plus respectively. This issue can be easily solved and does not require big system changes.

4.4 Conclusion

Despite the fact that there are two open issues, the goal of the research was achieved. The tourism application that mimics the functionality of the website without wireless connection, and consequently without costly Internet traffic and higher battery consumption, was developed, tested, reviewed and released. “FollowQTM” is a convenient app for people who enjoy agriculture tourism and want to support local farming. Since several quilt trails will be merged, the update functionality is vital for the idea of the application.

The success of the project was reached through the conducted research in this thesis. With the background work provided in the Review of Literature chapter, it was possible to create methods that became the foundation of the Follow the Quilt Trail Mobile project and the research of the thesis. The implementation phase was divided into two milestones. The first milestone required completing the core functionality of the application that included creation of the graphical user interface, developing the database and providing the core application logic. This part was successfully tested and released as V1.0. It has been available since September 2013 in the App Store. The target OS is iOS 6.1, but it does support the newer versions of the OS. The second milestone required the creation of client/server data synchronization logic. It included the development of both the client side logic written in Objective C/C and server side logic written in PHP. Both parts included the work with the databases: SQLite in the client side and MySQL in the server side. The communication between client and server was established through JSON serialization. This format is recognizable by both PHP and Objective C languages.

To conclude, the client/server data synchronization technique that was developed during thesis research helped to reduce the constraints of the web based application. Even though, the application is synchronized with the web site’s database, it can fully operate with no Internet access. As a result, it eliminates the problem of limited cellular coverage areas, higher energy consumption and Internet traffic usage.

References

Apple. (2014a, June). *Data management in ios.* Available from `https://developer.apple.com/technologies/ios/data-management.html`

Apple. (2014b, June). *Thread management in ios.* Available from `https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Multithreading/CreatingThreads/CreatingThreads.html`

AT&T. (2014, June). *Cell phone plans - cell phone, tablets device plans from at&t.* Available from `http://www.att.com/att/planner/#fbid=oNmXG6Xh1wE`

Dewey, C. (2013). Tourism goes mobile and social. (cover story). *Grand Rapids Business Journal*, *31*(18), 1. Available from `https://login.ezproxy.etsu.edu:3443/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=87527212&site=eds-live&scope=site`

Ding, N., Wagner, D., Chen, X., Pathak, A., Hu, Y. C., & Rice, A. (2013, June). Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. *SIGMETRICS Perform. Eval. Rev.*, *41*(1), 29–40. Available from `http://doi.acm.org/10.1145/2494232.2466586`

Fairbanks, A., Gribbons, J., Nybo, E., Pean, D., & Wright, J. (2002, May). Research in xml (extensible markup language). *J. Comput. Sci. Coll.*, *17*(6), 253–254. Available from `http://dl.acm.org/citation.cfm?id=775742.775785`

Garcia, F., & Fernandez, J. (2000, February). Posix thread libraries. *Linux J.*, *2000*(70es). Available from `http://dl.acm.org/citation.cfm?id=348120.348381`

Gil, B., & Trezentos, P. (2011). Impacts of data interchange formats on energy consumption and performance in smartphones. In *Proceedings of the 2011 workshop on open source and design of communication* (pp. 1–6). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/2016716.2016718`

Introduction to json. (2014, June). Available from `http://www.json.org`

Lin, D.-Y., & Neamtiu, I. (2009). Collateral evolution of applications and databases. In

Proceedings of the joint international and annual ercim workshops on principles of software evolution (iwpse) and software evolution (evol) workshops (pp. 31–40). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/1595808.1595817`

Oliver, E. A., & Keshav, S. (2011). An empirical approach to smartphone energy level prediction. In *Proceedings of the 13th international conference on ubiquitous computing* (pp. 345–354). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/2030112.2030159`

Sajid, A., Nayyar, A., & Mohsin, A. (2010). Modern trends towards requirement elicitation. In *Proceedings of the 2010 national software engineering conference* (pp. 9:1–9:10). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/1890810.1890819`

Sumaray, A., & Makki, S. K. (2012). A comparison of data serialization formats for optimal efficiency on a mobile platform. In *Proceedings of the 6th international conference on ubiquitous information management and communication* (pp. 48:1–48:6). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/2184751.2184810`

APPENDICES

Appendix A: Use Cases

Use Case # 1	View Barns on Map	
Goal in Context	Allow for a user to select county to see all the Barns on map by county	
Scope and Level	N/A	
Preconditions	Follow the Quilt Trail is run and user selects Select County on the main menu	
Success End Condition	ser finds the county he/she wants	
Failed End Condition	ser does not find the county he wants	
Primary, Secondary Actors	User, Data Store	
Trigger	User decides to view Barns on Map	
Main Success Scenario	Step	Action
	1	User selects county
	2	The system runs Select Barn on Map
Extensions	Step	Branching Action

Open Issues:

1.

Owner	Dmitry Tumanov	
Initial Entry Date	2/08/2013	
	Modification History	
Date	By Whom	Modification

Use Case # 2	Select Barn on Map	
Goal in Context	Allow for a user to select particular barn from barns available for particular county on Map Kit	
Scope and Level	N/A	
Preconditions	User selected particular county	
Success End Condition	User finds the barn he wants to get info about	
Failed End Condition	User cannot find the barn he wants,to get info about	
Primary, Secondary Actors	User, Data Store, Map Kit	
Trigger	User decides to get information,about particular barn on map	
Main Success Scenario	Step	Action
	1	The system shows Map Kit with,all available barns for selected county
	2	User chooses barn on map
	3	System shows barn options
Extensions	Step	Branching Action
	3a1	System shows information about selected barn
	3b1	User chooses Get Directions
	3b2	Map Kit shows Direction

Open Issues:

1.

Owner	Dmitry Tumanov	
Initial Entry Date	2/08/2013	
Modification History		
Date	By Whom	Modification

Use Case # 3	Select Barn from list	
Goal in Context	Allow for user to select a particular barn from a list of barns	
Scope and Level	N/A	
Preconditions	User selected particular county	
Success End Condition	User finds the barn he wants to get info about	
Failed End Condition	User cannot find the barn he wants to visit	
Primary, Secondary Actors	User, Data Store	
Trigger	User decides to receive information about particular barn	
Main Success Scenario	Step	Action
	1	The system shows the list of barns ordered by county
	2	User selects the barn from the Barn List
	3	System shows barn options
Extensions	Step	Branching Action
	3a1	System shows information about selected barn
	3b1	System shows option show on map
	3c1	User selects add to favorite
	3b2	System stores barn ID on local data store

Open Issues:

1.

Owner	Dmitry Tumanov	
Initial Entry Date	2/08/2013	
Modification History		
Date	By Whom	Modification

Use Case # 4	Show Favorites	
Goal in Context	Allow for user to select a,particular barn from a list of favorite barns	
Scope and Level	N/A	
Preconditions	User selected Show Favorites	
Success End Condition	User finds the barn he wants to get info about	
Failed End Condition	User cannot find the barn he wants to visit	
Primary, Secondary Actors	User, Data Store	
Trigger	User decides to receive information about particular barn from Favorites list	
Main Success Scenario	Step	Action
	1	User selects the barn from the Favorites List
	2	System shows barn options
Extensions	Step	Branching Action
	2a1	System shows information about selected barn
	2b1	System shows option show on map

Open Issues:

1.

Owner	Dmitry Tumanov	
Initial Entry Date	2/08/2013	
	Modification History	
Date	By Whom	Modification

Use Case # 5	Update Barns	
Goal in Context	Allow user to receive database updates from the server	
Scope and Level	N/A	
Preconditions	User runs the app and his/her device has Internet access	
Success End Condition	User receives updates	
Failed End Condition	The data was not updated	
Primary, Secondary Actors	Updater, Data Store	
Trigger	New updates available in the server	
Main Success Scenario	Step	Action
	1	The system sends request for new updates to the server
	2	The server responds to the system
	3	The system populates local data store with received data
Extensions	Step	Branching Action
	2a1	The system updates existing record(s)
	2a2	The system deletes existing record(s)
	2a3	The system inserts new records

Open Issues:

1. Impossible to add new icon to the app's bundle

Owner	Dmitry Tumanov	
Initial Entry Date	05/05/2014	
Modification History		
Date	By Whom	Modification

Appendix B: "FollowQTM" Screenshots

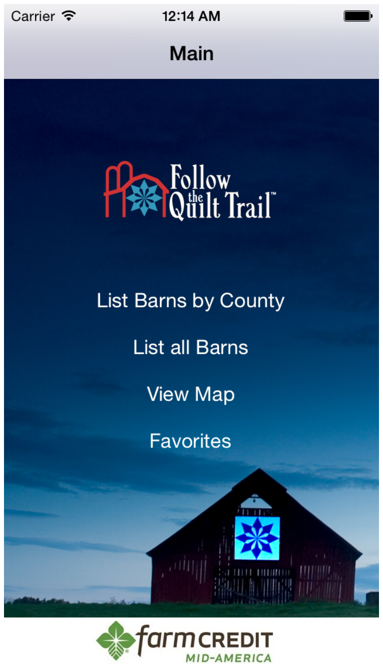
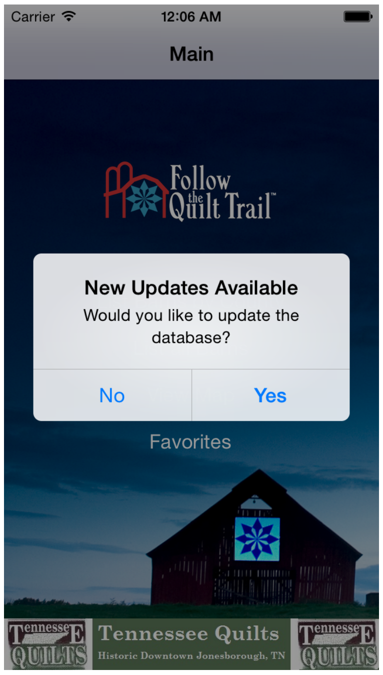


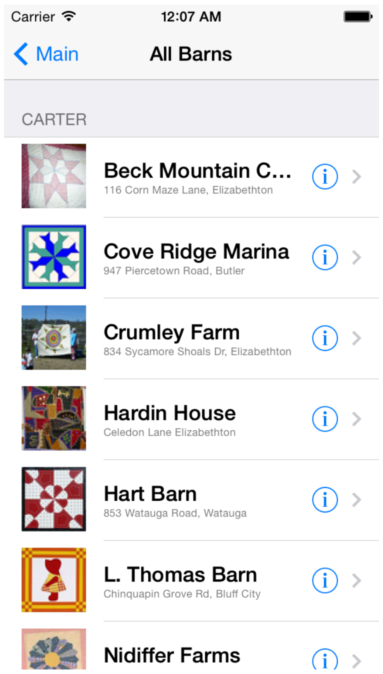
Figure 1: Main menu



New Updates Available

Would you like to update the database?

Figure 2: Update request



Beck Mountain C...
116 Corn Maze Lane, Elizabethton

Cove Ridge Marina
947 Piercetown Road, Butler

Crumley Farm
834 Sycamore Shoals Dr, Elizabethton

Hardin House
Celedon Lane Elizabethton

Hart Barn
853 Watauga Road, Watauga

L. Thomas Barn
Chinquapin Grove Rd, Bluff City

Nidiffer Farms

Figure 3: List of all barns

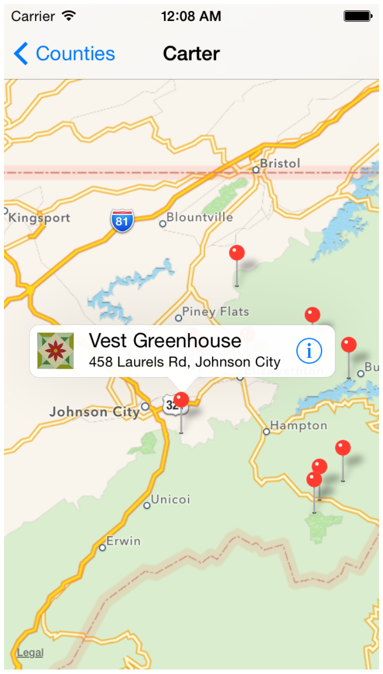
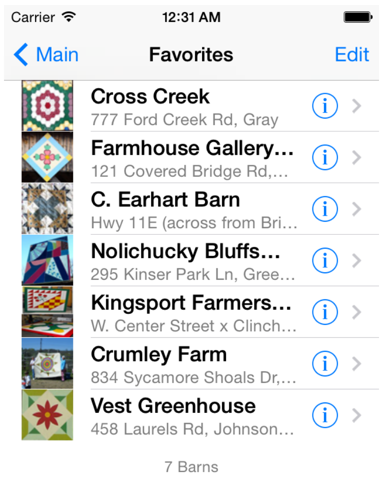


Figure 4: Map view



Cross Creek
777 Ford Creek Rd, Gray

Farmhouse Gallery…
121 Covered Bridge Rd,…

C. Earhart Barn
Hwy 11E (across from Bri…

Nolichucky Bluffs…
295 Kinser Park Ln, Gree…

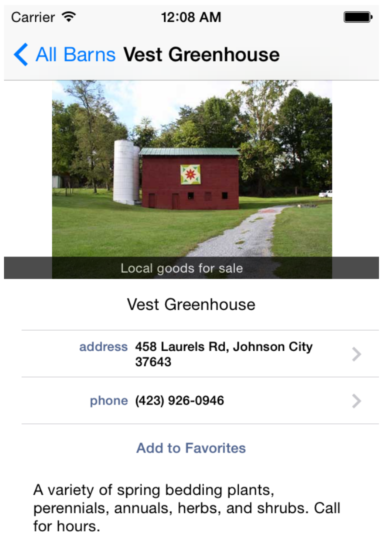
Kingsport Farmers…
W. Center Street x Clinch…

Crumley Farm
834 Sycamore Shoals Dr,…

Vest Greenhouse
458 Laurels Rd, Johnson…

7 Barns

Figure 5: Favorites



Vest Greenhouse

address 458 Laurels Rd, Johnson City 37643

phone (423) 926-0946

Add to Favorites

A variety of spring bedding plants, perennials, annuals, herbs, and shrubs. Call for hours.

Figure 6: Detail view

Appendix C: Communication Schema

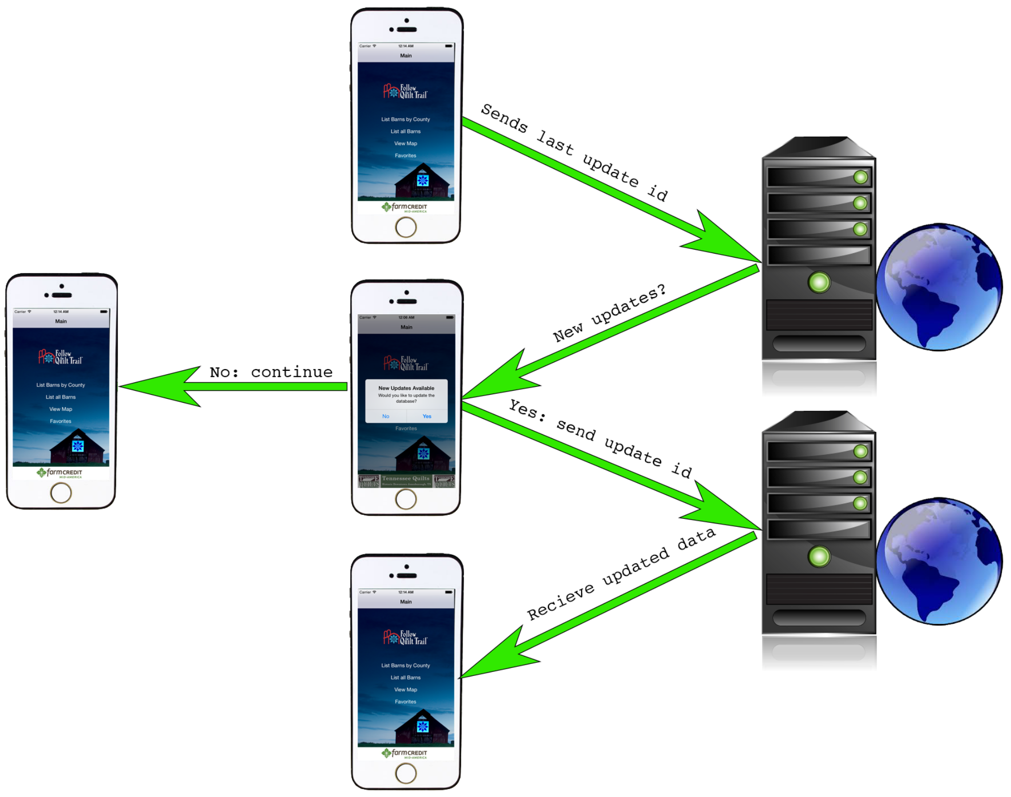


Figure 7: Communication schema