jamk.fi

ASYNCHRONOUS FILE TRANSFER BETWEEN iOS AND CLOUD

Teemu Laukka

Thesis

April 2014

Degree programme in Software Engineering

Technology, communication and transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s): LAUKKA, Teemu	Type of publication Bachelor's thesis	Date 21.04.2014
	Pages 27	Language English
	Confidential ( ) Until	Permission for web publications: (x)
Title ASYNCHRONOUS FILE TRANSFER BETWEEN iOS AND CLOUD		
Degree programme Software Engineering		
Tutor(s) MIESKOLAINEN, Matti		
Assigned by Nestronite Oy		

Abstract

The objective was to design and implement a native file transfer solution for Apple's iOS devices. The file transfer service was for an existing cross platform front-end application meant to collect user feedback in a form of media files. File data had to be send asynchronously from client to server to keep the application responsive to the user.

First the file transfer service architecture was designed and different technologies to implement it were evaluated. The service was programmed using the chosen technologies and coding conventions. The final step was to integrate the service with the existing front-end project and establish server interaction to send the file data to the cloud storage.

The thesis discusses basic iOS architecture and mobile development tools. Asynchronous file data transferring was achieved with the chosen technologies and it was tested to work with a real device and server environment.

Keywords
Apple, iOS, iPhone, file transfer, cloud storage

Miscellaneous

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t): LAUKKA, Teemu	Julkaisun laji Opinnäytetyö	Päivämäärä 21.04.2014
	Sivumäärä 27	Julkaisun kieli Englanti
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (x)
Työn nimi ASYNCHRONOUS FILE TRANSFER BETWEEN iOS AND CLOUD		
Koulutusohjelma Ohjelmistotekniikka		
Työn ohjaaja(t) MIESKOLAINEN, Matti		
Toimeksiantaja(t) Nestronite Oy		

Tiivistelmä

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa tiedostonsiirtojärjestelmä Applen iOS laitteille. Nestronite Oy:llä oli valmiina asiakaspuolen ohjelma liittyen käyttäjien palautteen keräykseen, johon iOS puolen ratkaisu tultaisiin yhdistämään. Tiedostot tuli lähettää pilvipalveluun taustatehtävänä ohjelmiston käyttökokemusta häiritsemättä.

Järjestelmän arkkitehtuuri suunniteltiin vaatimusten mukaan ja tarvittavat kehyskirjastot valittiin arvioinnin perusteella. Ohjelmakoodi kirjoitettiin ja dokumentoitiin sovittujen käytänteiden mukaan. Lopulta järjestelmä yhdistettiin asiakas- ja palvelinpuolen ohjelmistojen kanssa.

Opinnäytetyö selvittää iOS alustan arkkitehtuurin perusteet ohjelmistokehittäjän näkökulmasta, sekä kuinka iOS ohjelmia kehitetään. Tiedostonsiirtojärjestelmä toteutettiin vaatimukset täyttäen ja sen toiminta testattiin käytännössä oikeassa palvelinypäristössä.

Avainsanat (asiasanat)
Apple, iOS, iPhone, tiedoston siirto, pilvipalvelu

Muut tiedot

TABLE OF CONTENTS

FIGURES

TABLES

LIST OF ACRONYMS

API	Application Programming Interface, a set of routines and tools for building applications. An API specifies how software components should interact with each other.
BSD	Berkeley Software Distribution, a Unix operating system.
GCC	GNU Compiler Collection, a compiler system developed by the GNU Project.
HTTP	Hypertext Transfer Protocol, an application protocol used in the data communication in the World Wide Web.
OS	Operating System.
POSIX	Portable Operating System Interface, a set of standards for maintaining compatibility between operating systems.
SDK	Software Development Kit, a set of software development tools.
SOAP	Simple Object Access Protocol, a protocol for exchanging structured data in web services.
VoIP	Voice over Internet Protocol, a technology for delivering voice communications over Internet Protocol.

1 INTRODUCTION

1.1 Apple iOS

iOS, formerly known as iPhone OS, is Apple's mobile platform used in smart phones and tablets. The first version of iOS was availed in 2007 and then new version has been released every year. At the time of writing this, the newest iOS version 7 was released in September 18, 2013. It has over 20 percent market share of mobile devices being one of the major mobile operating systems with Google's Android and Microsoft's Windows Phone.

The iOS is derived from Apple's desktop operating system OS X. Both systems are based on Darwin operating system and they share some of the basic frameworks and technologies. iOS also has its own frameworks for more mobile features like multi-touch gestures and sensor interaction depending on the device. (iOS 2008.)

Darwin
Darwin is an open source POSIX-compliant operating system. It was released by Apple Inc. in 2000. The Darwin project is derived from various open source projects like NeXSTEP and BSD. Later it was used as the base of the Apple's OS X and iOS systems. (Darwin (operating system) 2002.)

1.2 Objective-C

OS X and iOS applications are mainly programmed with Objective-C language. It is a cross platform object oriented programming language originally developed in 1980s and used by the NeXSTEP project. Objective-C is a superset of the C-language which allows to mix C and Objective-C code in a certain extend. There is also a variant called Objective-C++, which in turns uses a combination of Objective-C and C++ syntax.

Objective-C extends C-language by adding support for classes and interfaces as well as more advanced features like key value coding, delegates and garbage collection mechanics. Objective-C programs are compiled with the GCC or Clang compiler. (Objective-C 2002.)

2 THE iOS SYSTEM

2.1 iOS Architecture

The iOS architecture consists of four main abstractions layers which progresses from higher level to lower level system frameworks. The higher level frameworks provides more object oriented abstractions from the lower level implementations. It is generally recommended to use top level frameworks that encapsulate all the complex features in a clean interface. That reduces the amount of code and keeps the code base easier to maintain. However, it is possible to use low level APIs to access some of the specific features which are not present in the high level frameworks. (About the iOS Technologies 2013.)

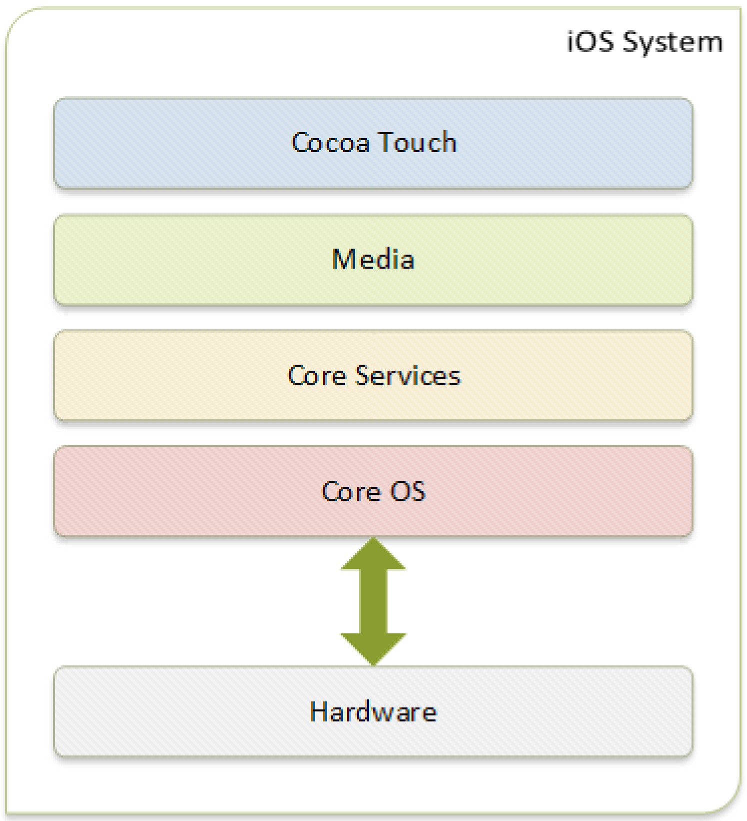

FIGURE 1. iOS layers

Core OS

The lowest layer, Core OS, is right at the top of the device hardware. It contains several services used to build other higher level technologies. Those services include Core Bluetooth framework to interact with different Bluetooth accessories, security frameworks for standardized security features and the System framework which contains C-based interfaces for low level memory and thread management. (Core OS Layer 2013.)

Core Services

The Core Services layer defines the basic types and core frameworks that every iOS application uses. These frameworks include iCloud storage services, location services, data storage technologies like SQLite and XML management and the Core Data framework unique to iOS. The most important ones are the Core Foundation framework and the Foundation framework which are same in iOS and OSX.

The Core Foundation framework provides C-based interfaces which include basic data management like collection data types and string management. The Foundation framework wraps most of the Core Foundation framework's features into Objective-C interfaces. These two frameworks can be mixed together with some degree. For example, some of the Core Foundation and Foundation types can be used in the functions of either framework. (Core Services Layer 2013.)

Media Layer

The Media layer contains necessary audio, video and graphics technologies to build various multimedia applications. Audio technologies include audio playing and recording capabilities as well as managing MIDI content. The video technologies support static video content and streaming videos from the Internet. On the graphical side, iOS has necessary high level UI frameworks for creating application views. It also supports lower level graphics APIs and technologies like OpenGL ES for advanced 2D or 3D rendering. (Media Layer 2013.)

Cocoa Touch Layer

The top level Cocoa Touch Layer frameworks define the look and feel of the application. When building an application, developers should first see, what these high level features provides before moving down on the layer hierarchy. Some of the high level features include multitasking model, Apple Push Notification service and storyboard interface designer. The Cocoa Touch layer also takes care of user's touch based input and gesture detection with different gesture recognizers. (Cocoa Touch Layer 2013.)

2.2 iOS Application development

iOS applications are developed using a Mac computer running the latest OSX version and Apple's Xcode development tools suite with the iOS software development kit. The main part of the suite is the Xcode integrated development environment. It contains fundamental code producing tools and features like

- projects for source code management
- code editor
- build system
- code compiler
- debugger
- static code analyzer
- device manager.

Other tools include an interface builder for constructing graphical user interfaces, different device simulators for testing the app and a direct access to Apple's developer library documentation also referred to in this document. In addition Xcode is needed for the application release process. (iOS Developer Tools 2013.)

It is possible to develop and test applications for free with Xcode IDE and iOS simulator, however, to develop on a real device requires Apple's developer program account. Apple has different enrollment options for individual developers and companies. Individual enrollment costs about 80 euros per year. For that sum of money the developer can develop and test the program with devices running iOS and distribute apps in the Apple's App Store service.

Native iOS applications are built using iOS frameworks included in the iOS SDK. The programming language is Objective-C or some of its dialects. iOS also supports web-based apps using HTML, CSS and JavaScript technologies. Unlike normal apps, which are physically installed on the device, web apps are not running directly on iOS. They use Safari web browser as their platform and so require network connection. Third option is to use a third party solution to make a hybrid app that can combine native and web based technologies. (About the iOS Technologies 2013.)

3 FEEDMON

3.1 Background

Feedmon is a mobile software developed by Nestronite Oy. The application is for collecting user feedback about products or services. The feedback can be given in three forms

- image
- audio recording
- video recording.

The user can, for example, buy a meal and tell his or her opinion on that in audio format. The restaurant can then use all those user reviews to improve their services and learn about the customers' likings. All review data are taken with mobile devices and the data has to be send to an external storage service over a network connection.

3.2 Technologies

The application uses various front- and back-end frameworks. All feedback media files are stored in the Amazon S3 cloud. The back-end is written in Node.js. Front-end uses PhoneGap mobile application framework and is implemented with common web technologies. The file uploading is handled via native implementation as a PhoneGap plug-in, thus being a hybrid type of application.

Transferring a file to a cloud is a two step process. First the file data is sent to a Node.js based server using multipart MIME technology and a REST interface. From there the data is finally moved to the Amazon S3 cloud storage.

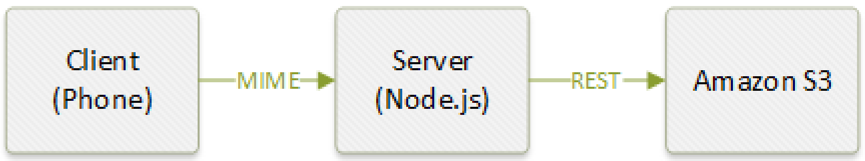

FIGURE 2. Feedback service architecture

3.2.1 Amazon S3 (Simple Storage Service)

Amazon S3 (Simple Storage Service) was first released in the United States in 2006, and a year after this it became available in Europe. The service provides online file storage space and supports REST, SOAP and BitTorrent technologies. In April 2013 there were more than 2 trillion objects stored in Amazon S3. Amazon guarantees 99.99% monthly up-time making it very reliable cloud storage system. (Amazon S3, 2013)

The objects stored in S3 are organized into buckets. Each individual object can be a size of a maximum of 5 terabytes. All the data is secured by the Amazon. The storage supports encryption and multiple access control mechanics. The buckets and objects can be accessed only by the resource owners. (Amazon S3 Product Details n.d.)

3.2.2 Node.js

Node.js software platform project was started by Ryan Dahl in 2009. The platform is used to build scalable network applications, mainly server side solutions. The IO is non-blocking by design and the event loop is single threaded. Node.js contains a built-in HTTP server library that makes it trivial to establish a working server side applications. (Nodejs 2009.)

Node.js platform is modular by design. It has a rather tiny core making it able to run well on a cloud infrastructure with fewer resources. The core Node.js is extended with its own package manager command line tool called npm (Node Package Modules). It makes updating and installing new node modules easy and keeps the project organized.

3.2.3 PhoneGap

PhoneGap is a cross platform mobile development framework owned by the Adobe Systems. It allows to implement hybrid mobile applications with popular web technologies

- HTML
- CSS
- JavaScript.

It is also possible to interface native components and develop native plug-ins that work seamlessly with the web side program. PhoneGap apps can be build locally or Adobe's PhoneGap Build service can be used. The build service consists of cloud compilers that, from uploaded source files, generates the app for supported platforms. PhoneGap supports all the major mobile platforms like Apple iOS, Google Android and Microsoft Windows phone. (PhoneGap 2009.)

3.2.4 REST

REST (Representational State Transfer) is a stateless architecture style first defined in 2000 by Roy Fielding. REST is based on HTTP 1.0 and it was developed in parallel with HTTP 1.1. The REST philosophy is mainly used for designing networking applications like web sites.

The REST architecture is meant to simplify client-server design by defining strict constraints. In general, clients do not know anything about the data storage behind the server. That improves the portability of the client software. On the other hand, the server side is not concerned about the user interface and state. This leads to a design where it is possible to replace clients and servers as long as they have a matching interface. (Representational state transfer 2004.)

The Representational State Transfer operates with request methods also included in the HTTP. The following table describes each of those methods.

TABLE 1. REST API methods

Method	Collection data	Element data
GET	List the details of the items in the collection.	Retrieve a member of the collection
PUT	Replace the collection with another collection.	Create or replace an existing member of the collection.
POST	Create new entry in the collection.	Not generally used. Can create a new entry to the member.
DELETE	Delete the entire collection.	Delete the member.

Multipurpose Internet Mail Extensions

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the original e-mail protocol SMPT (Simple Mail Transport Protocol). MIME allows for exchanging different kinds of data files like audio, video and image files through the network. This technique is tolerant to connection drops making it suitable for the file transferring between client and server. (MIME 2001.)

3.3 File transfer service

3.3.1 Requirements

Most parts of the existing PhoneGap program was cross platform, however, the native side file uploading code had to be implemented in Objective-C following common iOS guidelines. The file transferring had to function the way the operation does not interrupt the client program. The user can continue giving feedback regardless there is file exchange between client and server or not.

That kinds of requirements leads to utilizing some form of a background task system. The file uploading logic had to be designed so that it works under the strict time constraints.

3.3.2 iOS Background services

Sending a media file to the sever may take a significant amount of time to complete, therefore most of the uploading has to take place on the background, even when the user is not using the application. Only iOS 4 and higher supports that kind of background state and multitasking in general. The following figure shows different state changes in an iOS application life cycle. (App States and Multitasking 2013.)

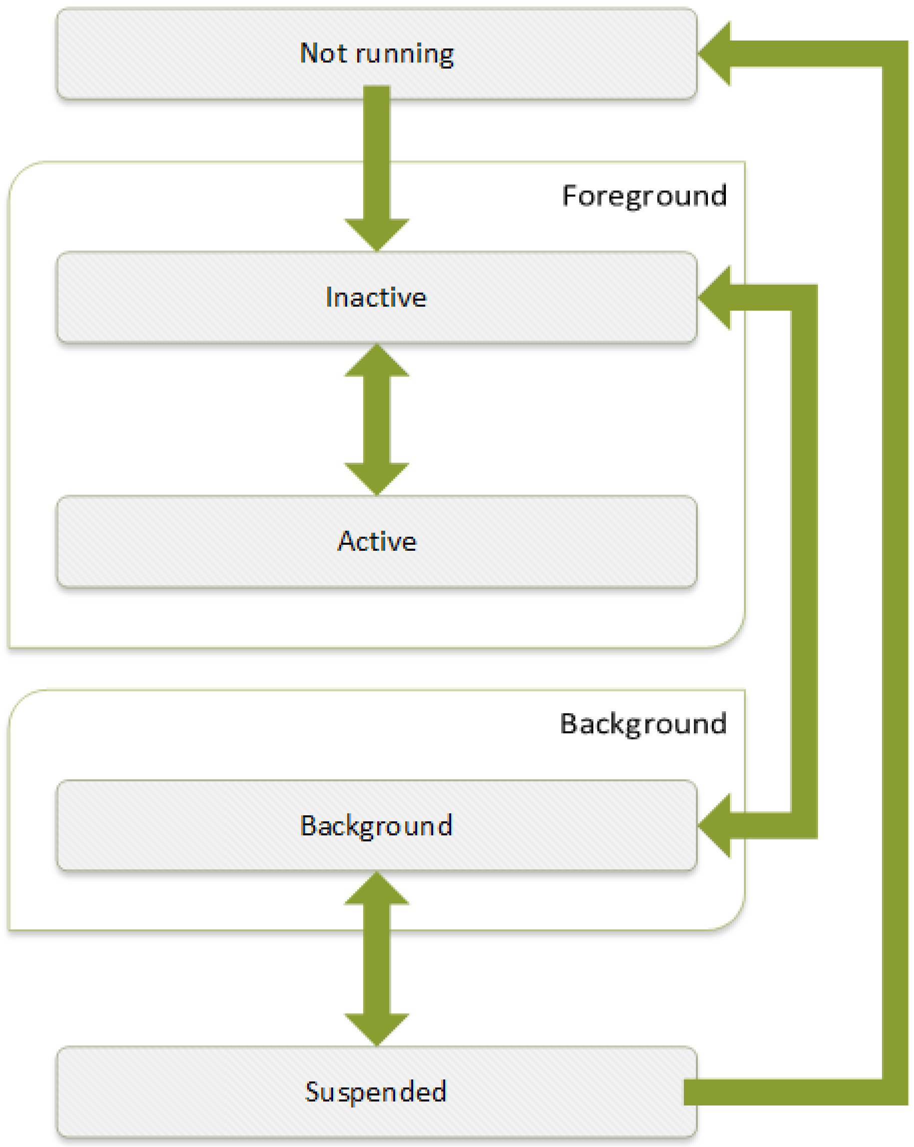

FIGURE 3. iOS application states

Not running

This is the initial state of the application. It has not been started or it is terminated by the system. Not running means the application is not loaded in the device memory and thus not executing any code.

Inactive

Usually an application stays briefly in inactive state before being active. In this state the app is running in the foreground but is not receiving events.

Active

Active state means the app is running normally on the foreground and receiving events.

Background

Apps that request extra time from the system can stay on the background state and finish a background task. Otherwise the app stays briefly on the background and then moves to the suspended state. An application can also be launched directly into the background state.

Suspended

In suspended state the app is in the background but is not executing any code. Suspended apps remain in the memory, however, the system may purge them to make more space for foreground apps. Purged apps return to the not running state.

3.4 Finite length background tasks

There are two types of background tasks in iOS system: finite-length and long running. Long running background tasks are only allowed for a certain type of applications, such as apps that play audio like music players, apps that support VoIP, and apps that receives regular updates from some external accessories. The file upload service does not meet any of these requirements, thus the appropriate choice is finite-length tasks. (App States and Multitasking 2013.)

When the app is about to move to the background, it can request additional time to complete some important finite-length task. During that extra time it is possible to run an operation such as file upload on the background. After the operation completes or the granted time runs out, the application moves to the suspended state.

4 DEVELOPMENT

4.1 Naming conventions

Apple recommended naming conventions were used in the program. Class names use camel case notation and are prefixed as FM- (as FeedMon, the project name) to indicate namespace and avoid possible naming conflicts with other framework classes.

4.2 Architecture

4.2.1 PhoneGap native iOS plug-in

The project contains two parts: the PhoneGap plug-in side and the actual file transfer service. The plug-in part consists of three media views: audio, camera and video view. Each of these views is responsible for capturing the respective media: audio clip, picture or video clip. Once the desired media is captured, the send service takes control.

The file data is sent to the server through multipart file upload that runs in the background thread. The result whether the operation succeeded or failed, is reported back to the plug-in bridge and finally to the client program.

4.2.2 AFNetworking

Standard iOS frameworks include some services for basic networking code. However, for more advanced implementations, there is a couple of third party networking libraries available. A popular iOS and OS X compatible networking library called AFNetworking is used to implement the send service.

AFNetworking is built on top of the URL Loading System of Apple's Foundation framework. It extends the built-in networking features found in the Cocoa framework. Those features include a thread management system for background operations and high level interaction with REST interfaces. (AFNetworking n.d.)

4.2.3 Asynchronous design

The file send service supports synchronous and asynchronous operations. In general, sending a relatively large file over the network is a slow process, therefore it makes the most sense to use asynchronous design. This leads to a more fluent user experience since upload processes are handled on the background and the main thread is not blocked keeping the app responsive.

Even when there are problems sending the file, network connection might be down or the server might not respond, the user must be able to continue to use the application as normal. For that reason, there has to be some mechanism to locally store unsent files in case issues arise.

4.2.4 Core Data framework

Apple's Core Data Framework is a way to store data in a structured object oriented way. While this system is not a relational database, it work in a similar fashion as an ORM implementation over an SQLite database. The Core Data supports object saving, loading and querying. (Core Data Programming Guide 2004.)

In this project Core Data was used to save information about the captured media files; this mainly for the purpose of keeping track of which files were sent to the server and which are still stored in the device. The used Core Data Model is illustrated in the following table:

TABLE 2. File send service Core Data model

Property name	Type	Description
path	String	Path to the media file in device storage
status	Integer	Media file status: sent or unsent
taskId	String	Identifier to connect the media file to the correct task
taskType	Integer	Type of the task: picture, audio or video
username	String	The user who captured the media file

Entries are never deleted, only the status field is updated. This keeps the history of the captured media files in the device.

4.2.5 Callback functionality

The file transfer service makes heavy use of callback mechanics to internally communicate with the different parts of the code. Callback functionality is achieved with Apple's C-level extension called block objects. This extension is only available for iOS 4.0 and newer.

Block objects are closure like function expressions that can be passed to other functions as a parameter, making them usable by multiple threads. With this technique, different block of code can be executed in different situations. For example, every operation related to the server traffic and local file data manipulation has a failing and a success block option. (Blocks 2011.)

4.2.6 Class structure

The main class structure of the send service is really simple as the following class diagram shows. The service contains three main classes which handles the three main tasks: capture, save and send. Additionally there are few smaller classes providing helper functions and encapsulating plain data fields to a single objects.

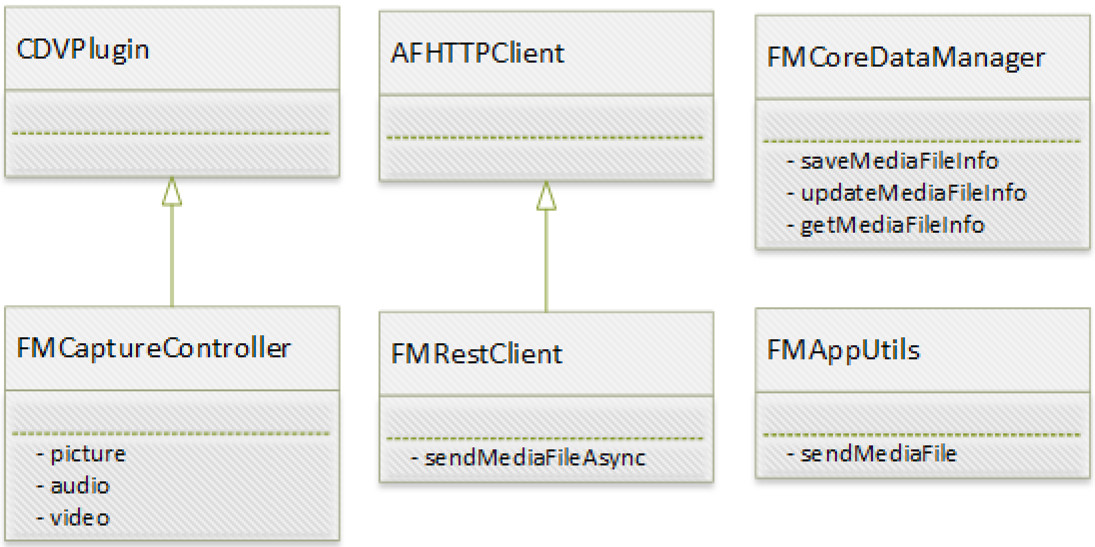

FIGURE 4. File send service class diagram

FMCaptureController

As a PhoneGap plug-in, this controller class is the bridge between the native and web side of the program implementing the iOS specific PhoneGap base class CDVPlugin. The controller is responsible for capturing the media files and preparing them for the further processing.

FMRestClient

The FMRestClient implements the file sending interface. It establishes a connection to the specified server and offers functions to asynchronously transfer file data from the client to the external storage. The rest client also manages a queue of unsent files in a separate thread.

FMCoreDataManager

The FMCoreDataManager class is in charge of local file data management. It provides a simple interface for manipulating the internal object data storage.

FMAppUtils

The main purpose of this class is to organize all the individual services and wire them together under a single function call. FMAppUtils attaches correct callback blocks to all Core Data and Rest client operations.

4.2.7 Application flow

In the application's normal behavior, it is assumed there is a working network connection and the server is available. The following sequence diagram shows the standard application flow when capturing a media file.

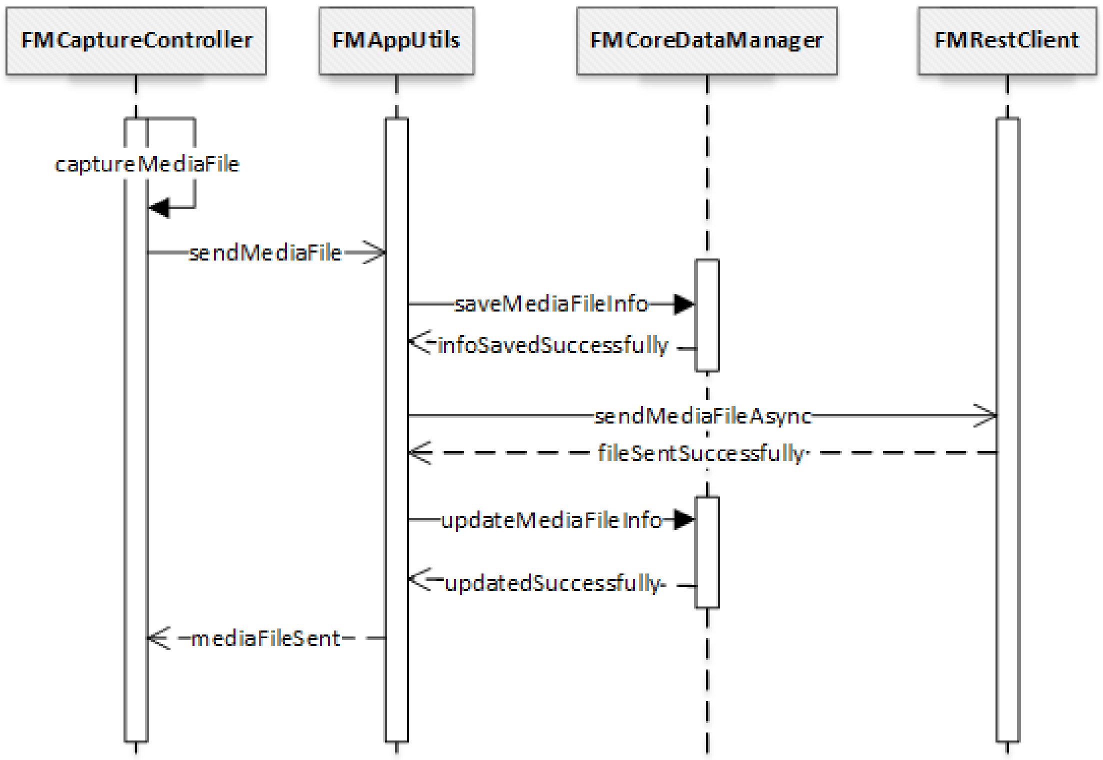

FIGURE 5: File transfer service sequence diagram

The program does not proceed to the next operation until the previous one is executed successfully. If problems occur during the file upload, the operation is put to a hold and the local copy of the file is kept on the device storage. After finishing the cycle, the service checks if there are more files on the queue and, if that is the case, starts the procedure again.

4.3 Testing

For testing iPhone simulator and an iPhone 5 device were used. Simulator versions 6 and 7 were merely used to test the application runs on a device, since simulators do not support media capturing like taking pictures. A real device was used to test that the actual application and file sending worked correctly.

The testing environment included the Node.js server locally running on the development machine and the Amazon S3 storage server of where the files were ultimately transferred.

4.4 Problems

Even though the front-end application was meant to be cross platform, it was optimized for Android platform. Some of the code structures between JavaScript and a native plug-in did not work in iOS environment. That bridge had to be modified so that it works with both platforms, Android and iOS.

Achieving a solid multithreaded asynchronous file transfer service was not easy. It took three iterations to find a correctly working architecture that suited the application specification. Completely custom service was first attempted to implement with only the standard iOS SDK. In the end, it turned out to be much reliable to use well tested AFNetworking framework to handle the critical parts of the networking code.

Large scale user testing is also troublesome with the iOS apps. Apple requires that every device used in testing has to be authorized separately and the test version of the app has to be shared through the iTunes (Apple's mobile device management application). Every tester need to send their device ID to the application owner, which might be time consuming. (Beta Testing Your iOS App 2014.)

4.5 Improvements

The file send services is designed mainly for the Feedmon app to handle specific type of file data. Generally it would be more valuable to keep the service interface as generic as possible. The more generic service would allow it to be used in a different kinds of situations where file data has to be sent to a server over the Internet. The service does not necessarily have to know the details of the data to be send, only its type.

Also separating the PhoneGap plug-in side from the file transfer side would make the services more portable. The file transfer service does not have to depend on the PhoneGap specific implementations. It should act as a self contained entity that can be even compiled to a library.

5 RESULTS

Asynchronous file transferring between iOS and cloud was achieved. The file transfer service was implemented according to the company's needs following the agreed working methods. The specified requirements were taken into account during the development.

The working service was completed on a fairly tight schedule. Unfortunately, due to the pressure getting the Android version of the application to work and released, the iOS version did not make it into the pilot stage. Because of the Apple's strict app release procedures, it will take some resources of the startup company to setup the developer licenses and application certificates and arrange proper beta testing.

The project gave a good hands-on introduction to iOS development. There were many difficulties that had to be solved concerning the asynchronous application design in mobile platform. Knowledge in advanced topics like network coding and thread management were vastly increased during the development.

It was an interesting experience to work in a small startup company where one has to adapt to flexible working methods and be prepared for rapid changes of the requirements of the application.

REFERENCES

iOS. 2008. Wikipedia article. Accessed on 7 October 2013. Retrieved from http://en.wikipedia.org/wiki/IOS

Darwin (operating system). 2002. Wikipedia article. Accessed on 7 October 2013. Retrieved from http://en.wikipedia.org/wiki/Darwin_(operating_system)

Objective-C. 2002. Wikipedia article. Accessed on 7 October 2013. Retrieved from http://en.wikipedia.org/wiki/Objective-C

About the iOS Technologies. 2013. Page on the Apple developer website. Accessed on 8 October 2013. Retrieved from https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html

Core OS Layer. 2013. Page on the Apple developer website. Accessed on 8 October 2013. Retrieved from https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/CoreOSLayer/CoreOSLayer.html#//apple_ref/doc/uid/TP40007898-CH11-SW1

Core Services Layer. 2013. Page on the Apple developer website. Accessed on 8 October 2013. Retrieved from https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/CoreServicesLayer/CoreServicesLayer.html#//apple_ref/doc/uid/TP40007

Media Layer. 2013. Page on the Apple developer website. Accessed on 8 October 2013. Retrieved from https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/MediaLayer/MediaLayer.html#//apple_ref/doc/uid/TP40007898-CH9-SW4

Cocoa Touch Layer. 2013. Page on the Apple developer website. Accessed on 8 October 2013. Retrieved from https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html#//apple_ref/doc/ui

iOS Developer Tools. 2013. Page on the Apple developer website. Accessed on 8 October 2013. Retrieved from https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/iPhoneOSDeveloperTools/iPhoneOSDeveloperTools.html#//apple_ref/doc/uid/TP40007898-CH7-SW1

Amazon S3. 2006. Wikipedia article. Accessed on 14 October 2013. Retrieved from http://en.wikipedia.org/wiki/Amazon_S3

Node.js. 2009. Wikipedia article. Accessed on 14 October 2013. Retrieved from http://en.wikipedia.org/wiki/Nodejs

PhoneGap. 2009. Wikipedia article. Accessed on 14 October 2013. Retrieved from http://en.wikipedia.org/wiki/PhoneGap

App States and Multitasking. 2013. Page on the Apple developer website. Accessed on 3 March 2014. Retrieved from https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html

AFNetworking. N.d. Page on the Github website. Accessed on 22 October 2013. Retrieved from https://github.com/AFNetworking/AFNetworking

Blocks. 2011. Page on the Apple developer website. Accessed on 22 October 2013. Retrieved from https://developer.apple.com/library/ios/documentation/cocoa/Conceptual/Blocks/Articles/00_Introduction.html

Amazon S3 Product Details. N.d. Page on the Amazon website. Accessed on 9 April 2014. Retrieved from http://aws.amazon.com/s3/details/

Representational state transfer. 2004. Wikipedia article. Accessed on 9 April 2014. Retrieved from http://en.wikipedia.org/wiki/Representational_state_transfer

Beta Testing Your iOS App. 2014. Page on the Apple developer website. Accessed on 9 April 2014. Retrieved from https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/TestingYouriOSApp/TestingYouriOSApp.html

Core Data Programming Guide. 2004. Page on the Apple developer website. Accessed on 9 April 2014. Retrieved from https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreData/Articles/cdTechnologyOverview.html#//apple_ref/doc/uid/TP40009296-SW1

MIME. 2001. Wikipedia article. Accessed on 9 April 2014. Retrieved from http://en.wikipedia.org/wiki/MIME