

TV SCHEDULES APP FOR IPHONE/IPOD TOUCH

by

ANUPAM GODBOLE

B.E, University of Pune, 2006

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2009

Approved by:

Major Professor
Dr. Daniel Andresen

Abstract

TV Schedules App for iPhone/iPod Touch is an interactive App that helps users to keep up-to-date with favorite TV Shows by displaying the airing time and date of the upcoming episode.

The primary focus was to get familiar with Cocoa Touch framework and Objective-C programming language to provide an intuitive GUI to enable users to add and remove favorite TV shows from a list. The App not only provides an easy and convenient way to keep track of the upcoming episodes but also allows the users to be notified minutes before the upcoming episode is about to be aired.

Table of Contents

List of Figures	v
List of Tables	vi
Acknowledgements	vii
Dedication	viii
CHAPTER 1 - Introduction	1
1.1 Motivation.....	1
1.2 Scope.....	1
1.3 Goal.....	1
1.4 Assumptions.....	1
1.5 Platform Specifications.....	2
CHAPTER 2 - Developer Platform	3
2.1 Objective-C.....	3
2.2 Cocoa Touch Framework.....	3
CHAPTER 3 - Technologies	4
3.1 Tools and Technologies	4
3.1.1 Objective-C	4
3.1.2 Cocoa Touch Framework.....	4
3.1.3 Xcode	6
3.1.4 eXtensible Markup Language	7
CHAPTER 4 - Introduction to iPhone/iPod Touch App Development.....	8
4.1 Getting started.....	8
4.2 Running the App in iPhone Simulator	8
4.3 Running the App on iPhone/iPod Touch	8
CHAPTER 5 - System Architecture	10
5.1 System Architecture.....	10
5.2 Architecture of TV Schedules App.....	10
5.2.1 Client (Fat Client)	11
5.2.1.1 Main Screen	11

5.2.1.2 Add Screen.....	11
5.2.1.3 Episode Info screen.....	12
5.2.1.4 Series Info screen.....	12
5.2.1.5 Settings screen	12
5.2.2 Thin Server(thetvdb.com and tvrage.com)	13
5.3 Use Case Diagram	13
CHAPTER 6 - Testing.....	18
6.1 Performance Testing.....	18
6.2 Black Box Testing	19
6.3 Screenshots	20
CHAPTER 7 - Project Metrics	24
7.1 Project Metrics	24
CHAPTER 8 - Conclusion and Future Work	25
8.1 Conclusion	25
8.2 Future Work.....	25
References.....	27

List of Figures

Figure 3.1 Cocoa Touch Framework Architecture	4
Figure 3.2 How Xcode uses source file references, targets, and executable environments.	6
Figure 5.1 Two-tier Architecture	10
Figure 5.2 Use Case Diagram of TV Schedules App	13
Figure 5.3 Partial Class Diagram of TV Schedules App (Class Diagram 1).....	14
Figure 5.4 Partial Class Diagram of TV Schedules App (Class Diagram 2).....	16
Figure 6.1 CPU and Memory usage on iPhone Simulator (Run 1 with WiFi connectivity, Second run with simulated GPRS connectivity)	18
Figure 6.2 CPU and Memory usage on iPod Touch (With WiFi connectivity)	18
Figure 6.3 Main Screen of TV Schedules App in Portrait Mode.....	20
Figure 6.4 Main Screen of TV Schedules App in Landscape Mode	21
Figure 6.5 Episode Info screen	21
Figure 6.6 Series Info screen	22
Figure 6.7 Add Series Screen.....	22
Figure 6.8 Settings Screen	23

List of Tables

Table 7.1 Project Lines of Code	24
Table 7.2 Project Planning Phase.....	24

Acknowledgements

I would like to thank my Major Professor Dr. Daniel Andresen for his constant help, encouragement and guidance throughout the project. I especially acknowledge his help in suggesting new and innovative ideas for improving the overall user experience for the TV Schedules App.

I would also like to thank Dr. Gurdip Singh and Dr. Torben Amtoft for their support and for graciously accepting to serve on my committee.

Dedication

I would like to dedicate this project to my parents Mr. Ramesh Godbole and Mrs. Madhuri Godbole for their words of encouragement and helping me to get through the difficult times. I would also like to dedicate this project to my brother Mr. Atul Godbole for helping me solve some of the technical difficulties.

CHAPTER 1 - Introduction

1.1 Motivation

iPhone/iPod Touch has revolutionized the way internet is accessed on a mobile device. Both the devices use advanced technologies like touch screen, multitouch, accelerometer, GPS. The motivation to develop TV Schedules App comes from my urge to learn

Xcode, iPhone OS and Objective-C for building applications for iPhone/iPod Touch. The most important factor that motivated the development of the App was the ease of development and intuitiveness with which it can be developed.

Also, keeping track of the upcoming episodes of several TV shows at a time was a hassle. Hence a need was felt to develop an App that would show the airing time and date to the user, just by launching the App.

1.2 Scope

TV Schedules App can be used to add/delete TV shows to favorite list. This list gets updated at App startup and hence keeps the user up-to-date with the airing time/date of the upcoming episode of the favorite TV shows. Additionally, Google Alerts can also be set to inform the user minutes before the upcoming episode is about to be aired. Optionally, timezone correction can be applied to the airing time. The App also shows names of guest stars, directors, writers and ratings of already aired episodes.

1.3 Goal

Goal is to gain a good knowledge of the complete life cycle of the project development starting from the Requirement Gathering Phase to the Testing Phase. The implementation phase of this App also gives a hands-on experience in Objective-C, iPhone OS and the testing phase gives hands-on experience in use of tools like Instruments.

1.4 Assumptions

- User will have Internet Connection while using TV Schedules App.
- User will have iPhone/iPod Touch with iPhone OS 3.0 or above firmware.

- User will have a valid iTunes account to download TV Schedules.

1.5 Platform Specifications

- iPhone 2G/3G/3GS, iPod Touch 1G/2G/3G
- iPhone OS 3.0 or above firmware

CHAPTER 2 - Developer Platform

TV Schedules App for iPhone/iPod Touch is developed for the iPhone OS platform using the Cocoa Touch framework and Objective-C programming language. Xcode 3.2 is used as an Integrated Development Environment on Mac OS 10.6. The goal of this chapter is to give an overview of Objective-C programming language and the Cocoa Touch Framework.

2.1 Objective-C

The Objective-C language is a simple computer language designed to enable sophisticated object-oriented programming. Objective-C is defined as a small but powerful set of extensions to the standard ANSI C language. Its additions to C are mostly based on Smalltalk. Objective-C is designed to give C full object-oriented programming capabilities, and to do so in a simple and straightforward way.

2.2 Cocoa Touch Framework

Cocoa Touch is the application development environment for iPhone OS, respectively. Cocoa Touch include the Objective-C runtime and Cocoa Touch frameworks which includes Foundation and UIKit frameworks, is used for developing applications that run on iPhone OS. The Foundation framework implements the root class, NSObject, which defines basic object behavior. It implements classes that represent primitive types (for example, strings and numbers) and collections (for example, arrays and dictionaries). Foundation also provides facilities for internationalization, object persistence, file management, and XML processing. Its classes can be used to access underlying system entities and services, such as ports, threads, locks, and processes. Foundation is based on the Core Foundation framework, which publishes a procedural (ANSI C) interface. UIKit frameworks is used for developing an application's user interface. It includes classes for event handling, drawing, image-handling, text processing, typography, and inter-application data transfer. They also include user-interface elements such as table views, sliders, buttons, text fields, and alert dialogs.

CHAPTER 3 - Technologies

This chapter includes the details of the latest technologies and tools used to build this application, their benefits and implementation details. The chapter also describes the interaction of these tools with the Cocoa Touch Framework.

3.1 Tools and Technologies

The latest tools and technologies involved in building this App are: Objective-C, Cocoa Touch Framework, Xcode, eXtensible Markup Language.

3.1.1 Objective-C

The Objective-C language is a simple computer language designed to enable sophisticated object-oriented programming. Objective-C is defined as a small but powerful set of extensions to the standard ANSI C language. Its additions to C are mostly based on Smalltalk, one of the first object-oriented programming languages. Objective-C is designed to give C full object-oriented programming capabilities, and to do so in a simple and straightforward way.

Most object-oriented development environments consist of several parts:

- An object-oriented programming language
- A library of objects
- A suite of development tools
- A runtime environment

3.1.2 Cocoa Touch Framework

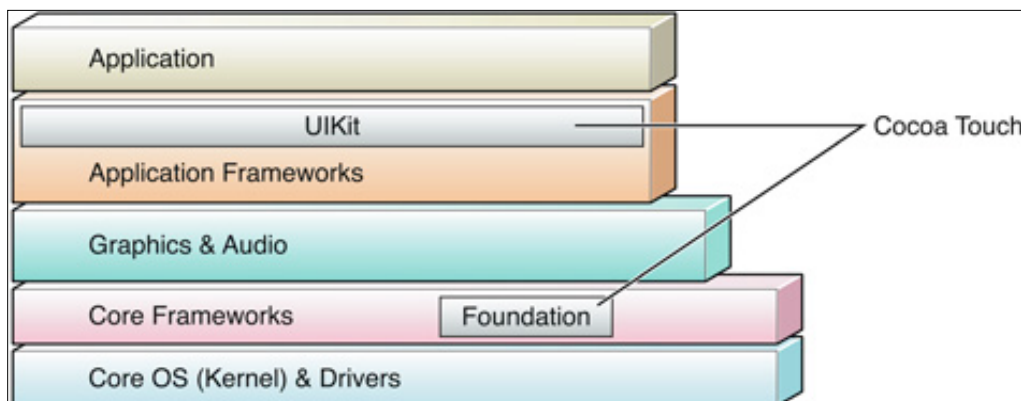


Figure 3.1 Cocoa Touch Framework Architecture

The following summarizes some of the frameworks found at each layer of the iPhone OS stack, starting from the foundation layer.

Core OS—This level contains the kernel, the file system, networking infrastructure, security, power management, and a number of device drivers. It also has the libSystem library, which supports the POSIX/BSD 4.4/C99 API specifications and includes system-level APIs for many services.

Core Services—The frameworks in this layer provide core services, such as string manipulation, collection management, networking, URL utilities, contact management, and preferences. This layer includes Core Foundation, a framework that provides abstractions for common data types (such as strings and collections); they allow a large degree of integration between object-oriented and procedural code. It also contains Core Data, a framework for object graph management and object persistence.

Media—The frameworks and services in this layer depend on the Core Services layer and provide graphical and multimedia services to the Cocoa Touch layer. They include Core Graphics, OpenGL ES, Core Animation, Core Audio, and video playback.

Cocoa Touch—The frameworks in this layer directly support applications based on iPhone OS. They include two Objective-C frameworks that are particularly important for developing applications for iPhone OS:

1. The UIKit framework provides the objects an application displays in its user interface and defines the structure for application behavior, including event handling and drawing.

The Foundation framework defines the basic behavior of objects, establishes mechanisms for their management, and provides objects for primitive data types, collections, and operating-system services. Foundation is essentially an object-oriented cover to the Core Foundation framework.

3.1.3 Xcode

Xcode is Apple's development environment for Mac OS X and iPhone OS. Xcode includes all the tools you need to create, debug, and optimize your applications. At the heart of the Xcode tools package is the Xcode IDE, a graphical workbench that tightly integrates a professional text editor, a robust build system, a debugger, and the powerful GCC compiler capable of targeting Intel and PowerPC regardless of host platform. Xcode is both easy to use, and yet powerful enough to build mobile iPhone OS applications. The complete iPhone OS developer tools chain is distributed as part of Xcode; these tools include Interface Builder, Instruments, Dashcode, the WebObjects framework, and the complete reference documentation, to name just a few.

All activity in Xcode, from creating and editing files to building and debugging applications, revolves around projects. Xcode projects organize and give access to all of the files and resources that are used to build a software product. Regardless of what you are building, Xcode manages three kinds of information to build your product:

Source file references that include source code, images, localized string files, data models, and more. Targets that define the products to build. A target organizes the files and instructions needed to build a product into a sequence of build actions that can be taken. Executable environments in which you can run and test a software product. An executable environment defines the program that should be used to run the product with. In many cases, this will be the product itself, but doesn't have to be. In addition, the executable environment defines any command-line arguments and environment variables which should be used.

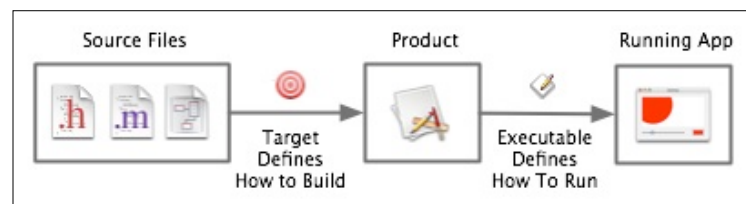


Figure 3.2 How Xcode uses source file references, targets, and executable environments.

When you execute the Build and Run command, Xcode processes a target that performs a set of actions on source code that in turn produces a product. Then, Xcode runs the product using the active executable environment.

3.1.4 eXtensible Markup Language

XML (eXtensible Markup Language) is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C and several other related specifications; all are fee-free open standards.

XML's design goals emphasize simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

There are a variety of programming interfaces which software developers may use to access XML data, and several schema systems designed to aid in the definition of XML-based languages.

CHAPTER 4 - Introduction to iPhone/iPod Touch App Development

4.1 Getting started

- The IDE for iPhone/iPod Touch App development is Xcode 3.x. It can be downloaded for free from Apple web-site <http://developer.apple.com>.
- Download the latest iPhone SDK from the same site.
- Install Xcode 3.x and iPhone SDK by following the instructions in the setup.
- Run Xcode 3.x and in the File Menu click on the “New Project...” sub-menu. In the dialog box that appears, click on the Application item under the iPhone OS heading in the left pane.
- Click on the Navigation-based Application and press the “Choose...” button. Provide a name for the App and you are ready to go. All the files needed to develop the App are created and the project is ready to build.

4.2 Running the App in iPhone Simulator

- Build the App by clicking on the “Build” sub-menu of the “Build” menu.
- Once the App is built with 0 errors, it can be run in the iPhone Simulator by clicking on one of the run commands in the “Run” menu.
- Doing so installs the App in the iPhone Simulator and runs it. Developer can debug the App by clicking on the “Debug” sub-menu in the “Run” menu.

4.3 Running the App on iPhone/iPod Touch

- To run the App on iPhone/iPod Touch, developer has to join the Apple iPhone Developer Program.
- Once joined, the developer can create profiles which is used to identify the developer, the iPhone/iPod Touch on which the App will be run and tested and the App to be run and tested. The whole thing is done thru developer.apple.com/iphone web portal. Once the profile is created on the portal, a file containing all the above information is generated by the portal which can be saved on the local drive.
- This file can be copied on to the iPhone/iPod Touch using iTunes or Xcode Organizer.

- Once that is done, select 'Device 3.0 – Debug' from the 'Project' → 'Set Active SDK' sub-menu.
- Build the App by clicking on the “Build” sub-menu of the “Build” menu.
- Once the App is built with 0 errors, it can be run in the iPhone/iPod Touch by clicking on one of the run commands in the “Run” menu.
- Doing so installs the App on the iPhone/iPod Touch and runs it. Developer can debug the App by clicking on the “Debug” sub-menu in the “Run” menu.

CHAPTER 5 - System Architecture

5.1 System Architecture

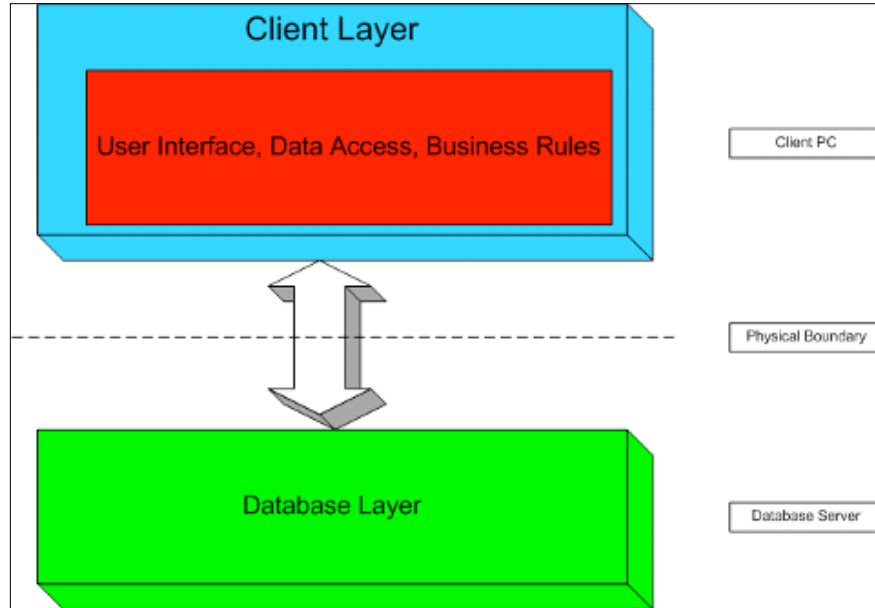


Figure 5.1 Two-tier Architecture

This chapter provides an Architectural Design for the TV Schedules App which represents a two tier architecture comprising of a client and a server. Since the business Logic is coupled with the Server, its called Fat Client with Thin Server. The client which in this case is the TV Schedules App running on an iPhone/iPod Touch is responsible for presenting the data retrieved from the server. thetvdb.com and tvrage.com are used as servers in this App to retrieve information about the TV series and its aired, upcoming episodes. An HTTP request is sent by the client to the server and the server responds to the request by sending relevant data in XML format as a response.

5.2 Architecture of TV Schedules App

TV Schedules App uses a two-tier architecture with a Fat Client i.e. most of the business logic is in the client and a thin server which just sends the information back to client on client's request.

5.2.1 Client (Fat Client)

TV Schedules App is comprised of multiple screens. Each one is described in details in the coming sections.

5.2.1.1 Main Screen

The class that represents the Main Screen has most of the business logic. Following chain of events happen when iPhone OS gives control to the TV Schedules App's class derived from UIApplication:

1. TV Schedules checks whether the App executable loaded is of current version or is an update. If it's an update, then it downloads the information of TV Series from the list all over again else it just downloads the information of the upcoming episodes of the TV Series.
2. User's favorite TV Series' upcoming episodes information is downloaded from the server tvrage.com by sending a HTTP request to it.
3. The main screen class waits for a response from the server which an XML file.
4. On receiving the response, it checks whether it's valid. If so, the response is parsed using xml parser library provided with iPhone SDK. Relevant information is retrieved, stored in the iPhone/iPod Touch's persistent memory and showed to the user on the main screen.
5. The summary of the upcoming episode is retrieved in a similar fashion from thetvdb.com server.

When the main screen is rotated left/right, the screen changes to a calendar view which shows the TV Series airing on the particular day.

5.2.1.2 Add Screen

On tapping the '+' button on the main screen, an 'Add Series' screen animates into the view. The series to be added to the favorite list can be entered in a textbox. Upon tapping the 'Search' button, TV Schedules App sends a request to the thetvdb.com server. The server sends an XML response as a list of TV Series containing the search keywords. This list is shown in a table view. User can select the TV Series to add to the favorite list by tapping on the TV Series name.

5.2.1.3 Episode Info screen

On tapping on the TV Series cell in the main screen, the App takes the user to a screen in which Summary, Director, First Aired and other details are shown. This info is obtained from the thetvdb.com server by sending it a HTTP request with the series ID, season & episode number. Info of the previously aired episodes is shown as well. The season number and episode number can be selected easily by tapping on the Previous, Next or Season Number and Episode number button.

5.2.1.4 Series Info screen

Tapping on the 'Series Info' button in Episode Info screen takes the user to the Series Info screen where it shows the series banner, Runtime, Content Rating etc. This info is obtained from the thetvdb.com server by sending it a HTTP request with the series ID.

5.2.1.5 Settings screen

Tapping on the gear like icon in the toolbar of the main screen brings the Settings screen. Various settings of the App can be changed from here. On tapping the done button, the changes are saved to the persistent memory. The various settings of the App are explained below:

- Timezone offset from EST: Determines the timezone in which the user is located so that corresponding offset can be added to the TV Series airing time.
- Overview in main window: Gives an option to the user to switch the visibility of summary in the main window.
- Series airing today badge: Option to show the number of series airing today badge on the App icon on the Home screen of iPhone/iPod Touch.
- Refresh at Startup: Determines whether the favorite TV Series list is updated when the App is launched.
- Refresh on Shake: An option to refresh the favorite TV Series list when the iPhone/iPod Touch is shaken.
- Google Alert reminder (mins): Number of minutes before upcoming episode's airing time that Google Alert sends a reminder.

5.2.2 Thin Server(*thetvdb.com* and *tvrage.com*)

thetvdb.com and tvrage.com servers just have the database which contains information about the TV Series, Episode Info and Series Info. They don't have any business logic other than accepting a request and sending a response. Both the servers expose a web-service which takes a HTTP request and gives a response in the form of a well-formed XML file. Following are some sample URLs.

- thetvdb.com
http://thetvdb.com/api/<DEVELOPER_KEY>/series/19345/default/2/1/en.xml
Retrieves the information of an episode of with TV Series id 19345, season number 2 and episode number 1.
- tvrage.com:
<http://services.tvrage.com/tools/quickinfo.php?show=House>
Retrieves the information about the upcoming episode of the TV Series House.

5.3 Use Case Diagram

Figure 4.3 shows the Use Case Diagram for the TV Schedules App. The user can Add Series to the favorites list, Remove Series from the list, View info of the aired/upcoming episodes of favorite TV Series and View favorite TV Series Info.

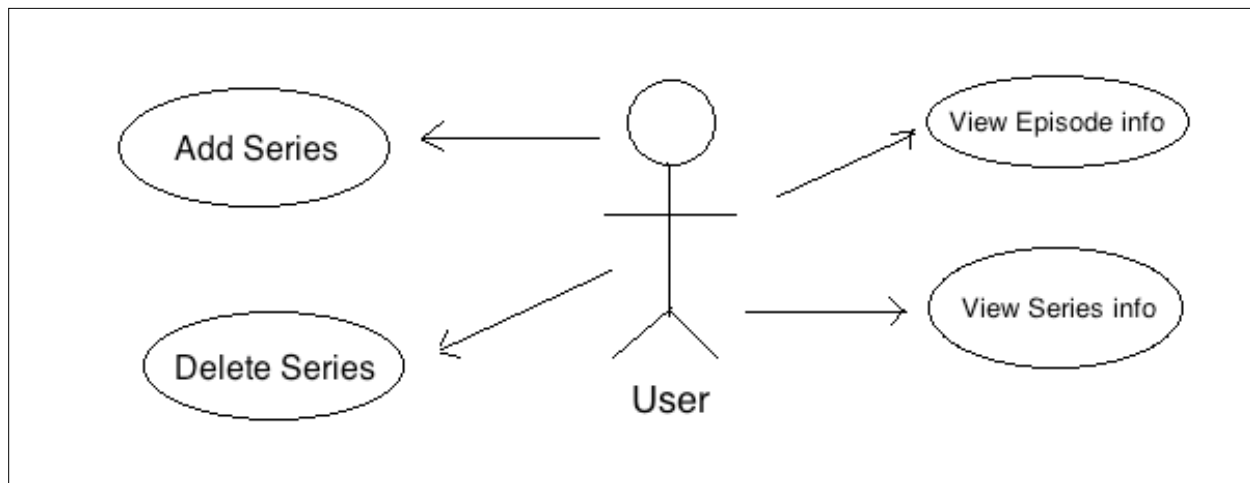


Figure 5.2 Use Case Diagram of TV Schedules App

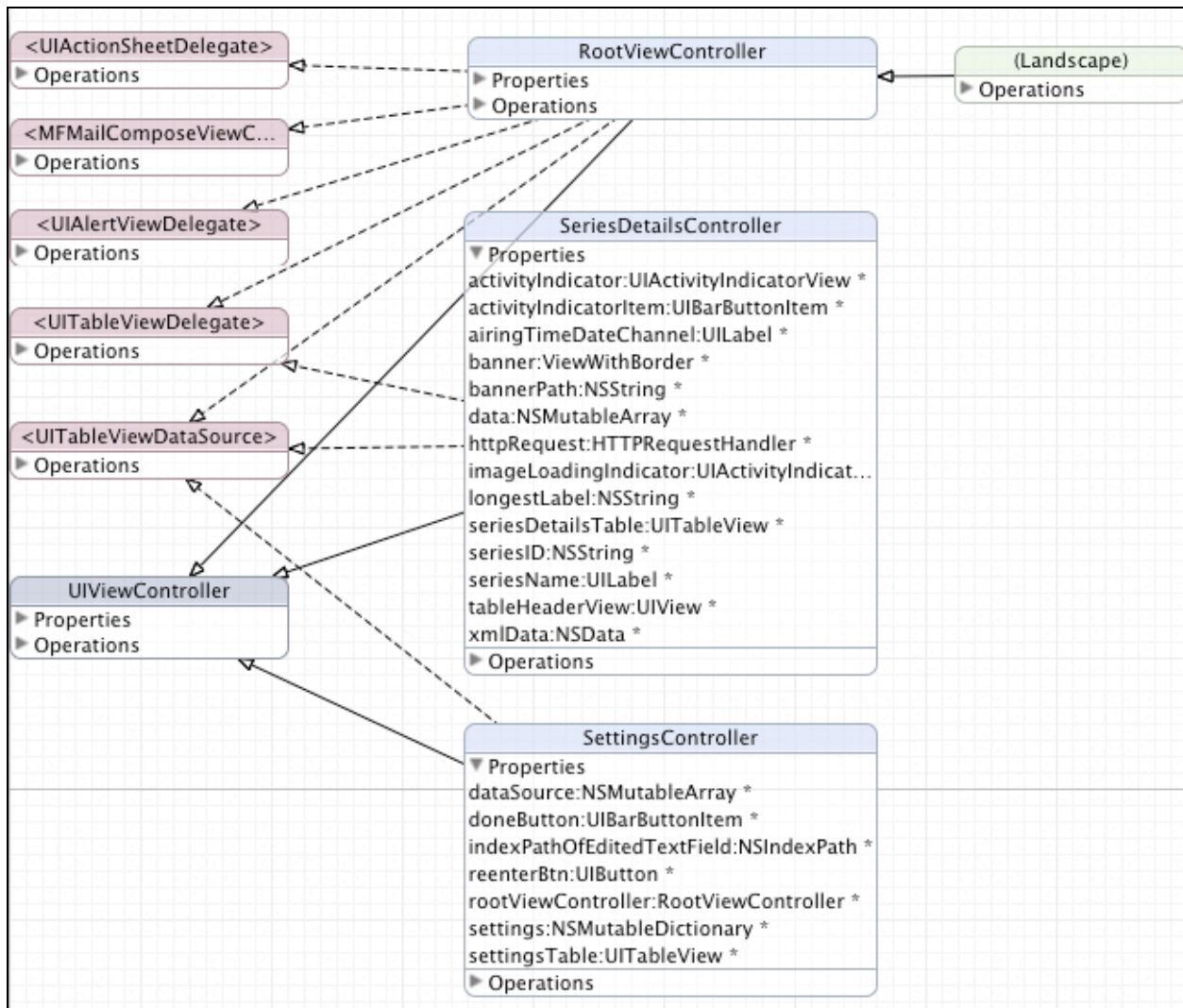


Figure 5.3 Partial Class Diagram of TV Schedules App (Class Diagram 1)

The classes and their role in the App architecture is explained below:

RootViewController class:

RootViewController class has UINavigationController as its base class. The UINavigationController class provides the fundamental view-management model for iPhone applications. The basic view controller class supports the presentation of an associated view in addition to basic support for managing modal views and rotating views in response to device orientation changes.

RootViewController class represents the “Main Screen” in the TV Schedules App. It is responsible for refreshing the upcoming episode's information by generating valid URLs,

accepting the response XML and parsing it to extract the required information. Libxml library provided with iPhone SDK is used for parsing XML. RootViewController is also responsible for handling events that get fired when the Calendar View is shown when iPhone/iPod Touch is in landscape mode.

SeriesDetailsController:

SeriesDetailsController class is also derived from UIViewController class. This class represents the “Series Info” screen in the App. Given a Series ID or Series Name, this class generates a valid URL, passes it on to the HTTPRequestHandler class, gets XML data from HTTPRequestHandler and finally parses it and extracts the relevant information.

SettingsController:

SeriesDetailsController class is also derived from UIViewController class. This class represents the “Settings” screen in the App. It loads the settings from the settings file and shows them as a graphical user interface. Whatever changes are made by the user are committed to the settings file on dismissing the screen.

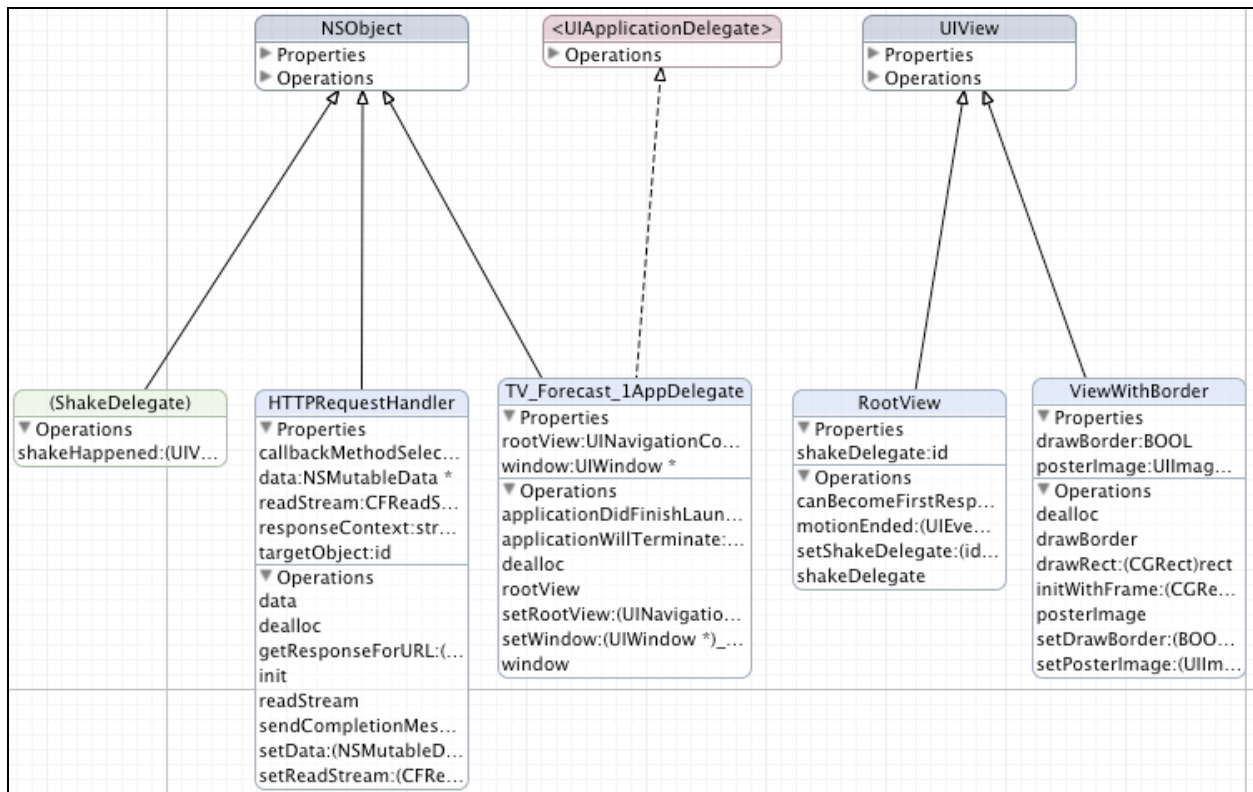


Figure 5.4 Partial Class Diagram of TV Schedules App (Class Diagram 2)

ShakeDelegate class:

ShakeDelegate class which is derived from NSObject is responsible for handling the shake event that is fired when iPhone/iPod Touch is shaken. When the event is fired, the class informs RootViewController class about it and RootViewController refreshes the favorite TV Series list in turn.

HTTPRequestHandler class:

This class takes a URL and a pointer to a function as an input. It opens a connection with the host, and requests the file specified in the URL. It schedules the request on a run loop which is kind of a thread that waits for the incoming response. When the runloop gets the response, it passes it on the function that's specified as an input.

TV_Forecast_1AppDelegate class:

This class handles the events that get fired when the App is loaded into the memory and control is transferred to the App. When the App is loaded, its `applicationDidFinishLaunching:` method is called. In this method an instance of the `RootViewController` class is created and shown as the “Main Screen”.

RootView class:

RootView class is derived from `UIView` class. The `UIView` class is primarily an abstract superclass that provides concrete subclasses with a structure for drawing and handling events. This class is responsible for laying out all the child `UIViews` in their proper position on initial load.

ViewWithBorder class:

ViewWithBorder class is also derived from `UIView` class. This class takes image data as an input, resizes it to the `UIView` size, draws a thin border around the image and finally renders it on the touch screen.

CHAPTER 6 - Testing

6.1 Performance Testing

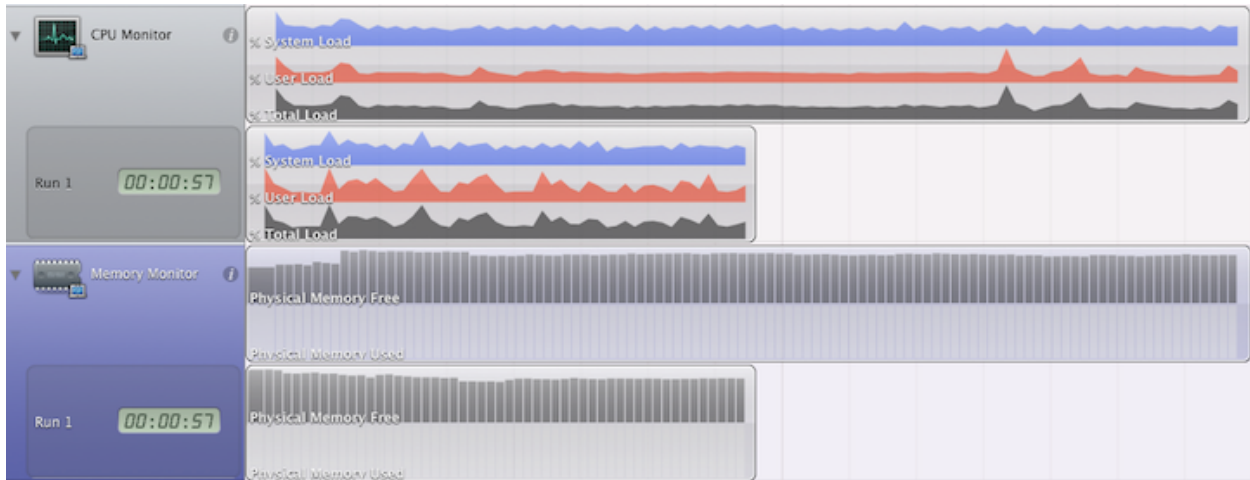


Figure 6.1 CPU and Memory usage on iPhone Simulator (Run 1 with WiFi connectivity, Second run with simulated GPRS connectivity)

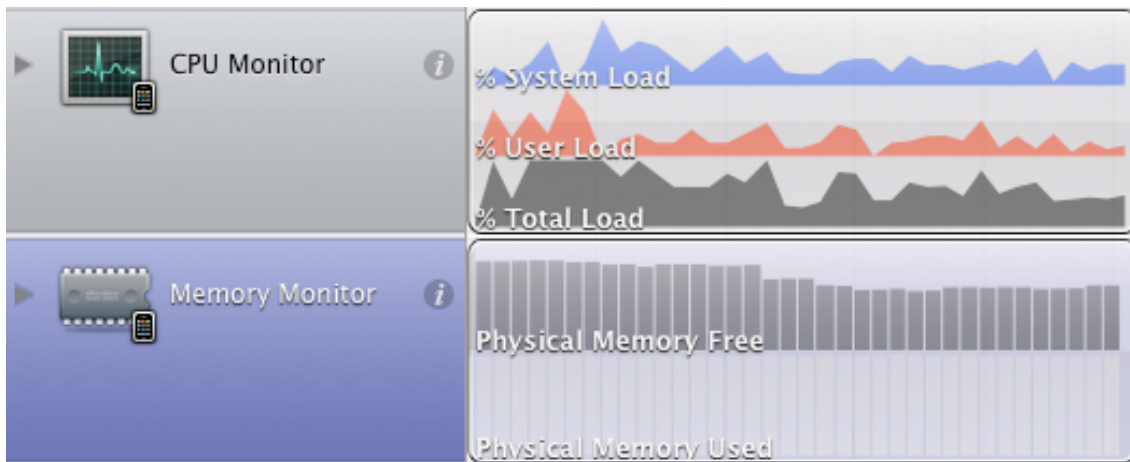


Figure 6.2 CPU and Memory usage on iPod Touch (With WiFi connectivity)

In Figure 6.1, the first run shows the System, User and Total Load when the App is run with Wi-Fi connectivity while the second run is with GPRS connectivity on an iPhone Simulator. As it can be seen the total load gets spread out in Run 2 as the data comes to App at a much slower rate and hence taking more time to perform the same task. As for the memory requirement, it remains unchanged in both the runs as can be seen in the figure. Figure 6.2 shows

the System, User and Total Load when the TV Schedules App is run on an iPhone/iPod Touch. As it can be seen, the nature of the graph is totally different as compared to those in Figure 6.1. Hence it's quite apparent that the performance of the App differs considerably on iPhone Simulator as compared to iPhone/iPod Touch.

6.2 Black Box Testing

Black box testing takes an external perspective of the test object to derive test cases. These tests check functional as well as non-functional aspects of the App. The test designer selects valid and invalid inputs and determines the correct output. Following cases were considered while doing the black box testing:

1. Low connectivity:

Low connectivity simulation isn't possible on iPhone/iPod Touch hence iPhone Simulator was used to achieve it. A software called “Speed Limit” was used for the purpose. Speed Limit limits the internet speed for the Mac OS X which in turn reduces the incoming and outgoing data rate for the iPhone Simulator. Because of low connectivity, the HTTP request timeouts randomly. Black Box Testing made it sure whether such a case was dealt with. On connection timeout or similar failure cases, TV Schedules App shows an error message informing the user about it.

2. Simulating packet loss:

A utility named ipfw was used to simulate packet loss. It randomly drops incoming and outgoing packets and hence simulating packet loss for iPhone Simulator. Because of the packet loss, connection timeouts randomly. In such a case, TV Schedules App shows an error message informing the user about it.

3. TV Series name with special characters in it:

To check whether special characters in the TV Series are processed correctly, various TV Series names like “The Office (US)”, “How clean is your house?” were used.

4. Ended/Cancelled series:

Since Ended/Cancelled series won't have any upcoming episodes. Asking the web service for its upcoming episode is just waste of bandwidth and hence it was made sure that that doesn't happen.

6.3 Screenshots

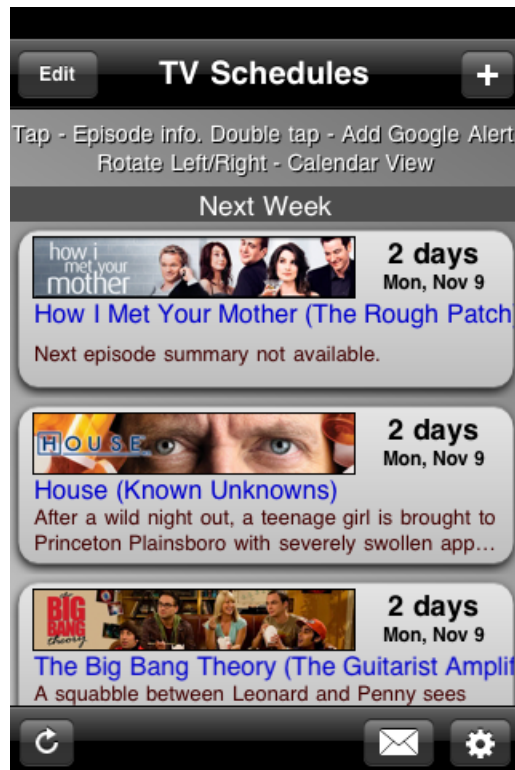


Figure 6.3 Main Screen of TV Schedules App in Portrait Mode

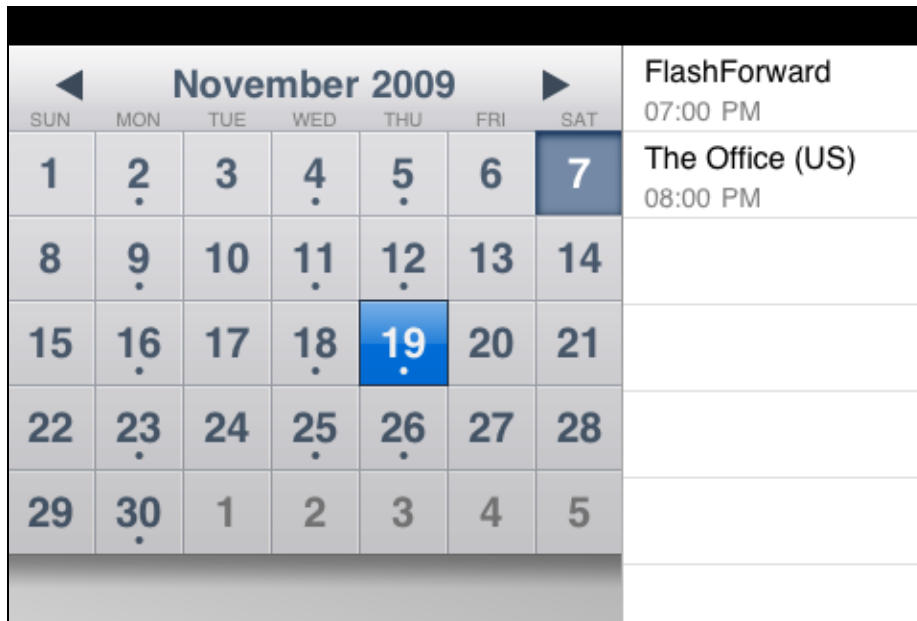


Figure 6.4 Main Screen of TV Schedules App in Landscape Mode

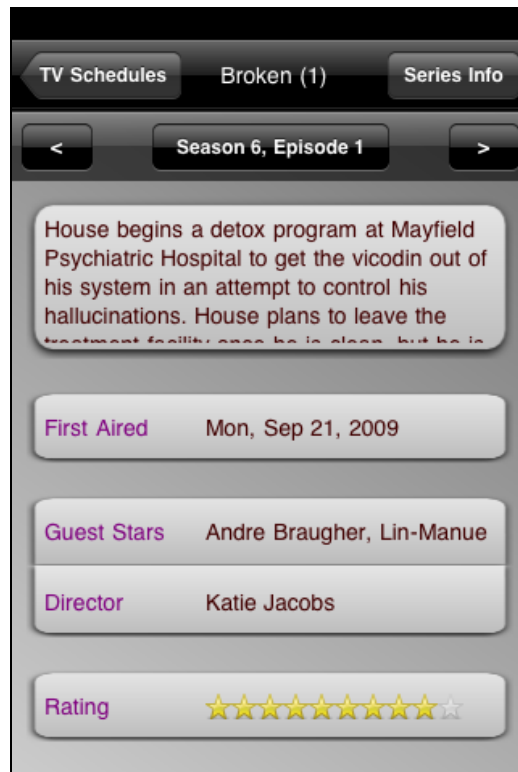


Figure 6.5 Episode Info screen



Figure 6.6 Series Info screen

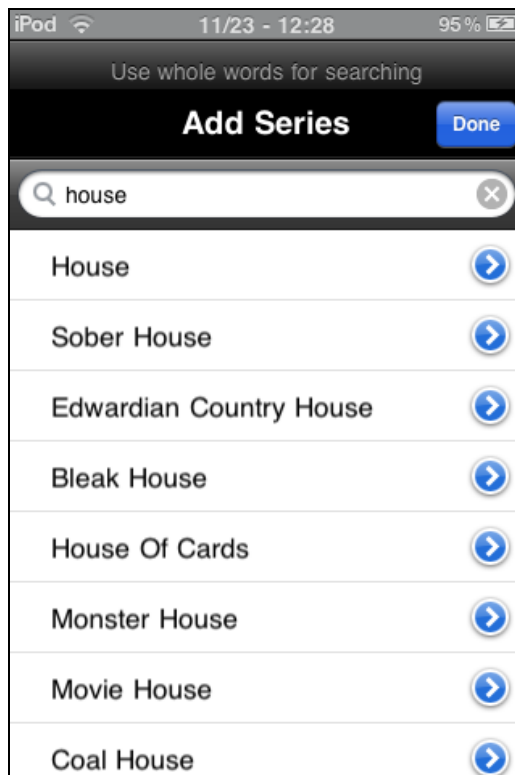


Figure 6.7 Add Series Screen

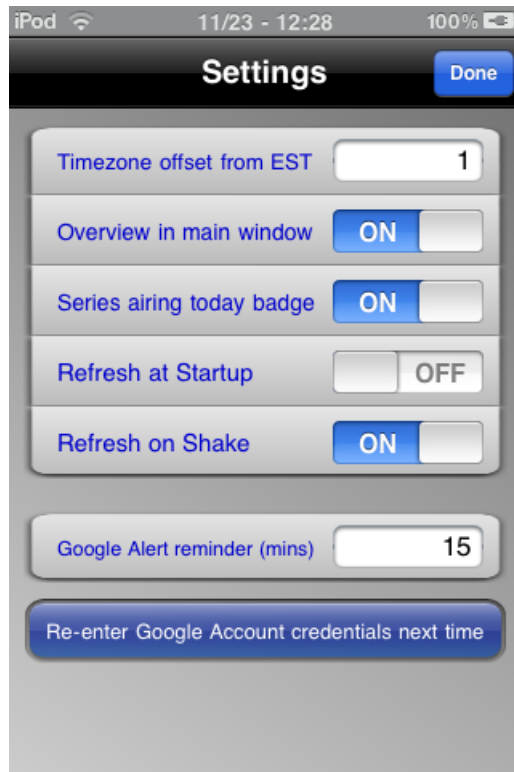


Figure 6.8 Settings Screen

CHAPTER 7 - Project Metrics

This chapter presents the project metrics showing the number of hours spent completing each phase of the project. It also summarizes the experiences gained during the entire life-cycle of the project.

7.1 Project Metrics

Project Metrics are the indicators that track the ongoing project progress. The Project Metrics discussed in this document are source lines of code and the amount of time spent during the entire project span. Table 1 and Table 2 represent the project metrics: source lines of code and project phases and their duration respectively.

Objective – C code	2500 lines
--------------------	------------

Table 7.1 Project Lines of Code

Learning Project Technologies	2 weeks
Requirement Gathering and Design	1 week
Implementation	10 weeks
Testing	2 weeks
Documentation	1 week

Table 7.2 Project Planning Phase

CHAPTER 8 - Conclusion and Future Work

This chapter describes the future scope and extensions for the project. There is still a huge scope of implementing something new and more to the project which can make it to the level of a commercial product.

8.1 Conclusion

TV Schedules App uses iPhone OS 3.0 SDK as a development platform and Xcode 3.2 as an Integrated Development Environment on Mac OS X 10.5.8 (Leopard). It uses two-tier architecture in which TV Schedules App acts as a fat client and thetvdb.com, tvrage.com act as thin servers. These servers accept the series ID, season number and episode number as a HTTP request and sends the response in XML format. This XML response is parsed using libxml library of the SDK, relevant information is extracted and displayed on the device. The whole user experience is made intuitive since iPhone/iPod Touch has advances hardware like touchscreen, accelerometer and powerful graphics processor.

8.2 Future Work

In the current iPhone OS framework, App aren't allowed to run in the background. Hence a user cannot be notified of airing of upcoming episode from iPhone/iPod Touch itself. TV Schedules App uses Google Alert service as a work around to solve the above problem. TV Schedules App adds a recurring event to the Google Calendar with the event time as the airing duration of the TV Series, event name as the TV Series Name, event location as the channel on which the TV Series is aired. If a user doesn't have a google account, this feature is unavailable to him. Secondly, user has to setup the alerts to be sent via email or SMS. User has to log into Google Calendar and specify the email address or a cell phone number for the same. This might seem to be a stretch for some users, hence reducing the usability of the App for those users.

Another work around that's less hassle for user is to use Apple Push Notification Service. The Apple Push Notification Service is a mobile service created by Apple Inc. that was released with iPhone OS 3.0. It uses push technology through a constantly-open IP connection to forward notifications from the servers of third party applications to the iPhone or iPod Touch; such notifications may include badges, sounds or custom text alerts. Implementing such a solution

would involve writing a server application and maintaining a database that contains the id of users' iPhone/iPod Touch and the TV Series present in their favorite TV Shows list. The server would go thru the database every few minutes and send a notification to the user about information of upcoming episodes few minutes before the airing time. Hence the need to have a Google Account to get the notifications would be done away with.

References

[1] iPhone Dev Center - Apple Developer Connection

<http://developer.apple.com/iphone>

[2] Stack Overflow

<http://www.stackoverflow.com>