EXPENSE TRACKER MOBILE APPLICATION

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Angad Manchanda

Fall 2012

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Angad Manchanda:

Expense Tracker Mobile Application

J. Carmelo Interlando

J. Carmelo Interlando, Chair
Department of Computer Science

Kris Stewart

Kris Stewart
Department of Computer Science

Joseph Lewis
Department of Computer Science

December 6, 2012

Approval Date

DEDICATION

I would like to dedicate this thesis to my dearest parents and my brother who believed and kept full faith in me and provided me the zeal that was required for the accomplishment of the thesis project.

ABSTRACT OF THE THESIS

Expense Tracker Mobile Application
by
Angad Manchanda
Master of Science in Computer Science
San Diego State University, 2012

Modern life offers a plethora of options of services and goods for consumers. As a result, people's expenses have gone up dramatically, e.g., compared to a decade ago, and the cost of living has been increasing day by day. Thus it becomes essential to keep a check on expenses in order to live a good life with a proper budget set up.

The iPhone device, designed and marketed by Apple Inc., is one of the top-selling smartphones in the USA, and with the launch of the new iPhone5 on September 21, 2012, whose sales have already surpassed the previous iPhone handsets (iPhone 4S, iPhone4) sales, it is apparent that people have been using smartphones as an organizational tool.

XpensTrak, the Expense Tracker Mobile Application was developed for iPhone users to keep track of their expenses and determine whether they are spending as per their set budget. Potential users need to input the required data such as the expense amount, merchant, category, and date when the expense was made. Optional data such as sub-category and extra notes about the expense can be entered as well. The application allows users to track their expenses daily, weekly, monthly, and yearly in terms of summary, bar graphs, and pie-charts.

This mobile application is a full detailed expense tracker tool that will not only help users keep a check on their expenses, but also cut down the unrequired expenses, and thus will help provide a responsible lifestyle.

An analysis comparing existing expense tracking software with the one being introduced is provided.

TABLE OF CONTENTS

PAGE

LIST OF FIGURES

PAGE

LIST OF ABBREVIATIONS

iOS	iPhone Operating System
IDE	Integrated Development Environment
SDK	Software Development Kit
Nib	Next Interface Builder
UI	User Interface
PDF	Portable Document Format
OS	Operating System
HTML	HyperText MarkUp Language
CSS	Cascading Style Sheets
SSL	Secure Socket Layer
WP7	Windows Phone 7
plist	Property List
PC	Personal Computer

ACKNOWLEDGEMENTS

I thank all the committee members for reading and evaluating my work.

I gladly take the opportunity to thank my thesis advisor, Prof. J. Carmelo Interlando, for the guidance, support, encouragement, and the motivation behind the success of my thesis. It would have been hard to accomplish this without his help.

I thank Dr. Joseph Lewis for providing me with invaluable suggestions and guidance on how to implement the functionality regarding the flow of the application and documentation of the thesis.

I thank Dr. Kris Stewart for her observations and invaluable comments.

CHAPTER 1

INTRODUCTION

With the launch and increase in sales of smartphones over the last few years, people are using mobile applications to get their work done, which makes their lives easier. Mobile applications comprise various different categories such as Entertainment, Sports, Lifestyle, Education, Games, Food and Drink, Health and Fitness, Finance, etc.

This Expense Tracker application falls in the Finance Category and serves the important purpose of managing finances which is a very important part of one's life.

The software product went through the design, development, and the testing phase as a part of the Software Development Lifecycle.

The application's interface is designed using custom art elements, the functionality is implemented using iOS SDK, and the phase of testing the product was accomplished successfully. The application is not much user intensive but just comprises of having them enter the expense amount, date, category, merchant and other optional attributes (taking picture of the receipts, entering notes about the expense, adding subcategories to the categories). With this entered information, the user is able to see the expense details daily, weekly, monthly, and yearly in figures, graphs, PDF format, and can print them as well if a printer is detected or scanned nearby. All these topics have been explained in detail in their respective chapters.

The aim of this thesis is to provide a solution for iPhone users on how to manage finances in any circumstance by keeping track of their expenses everyday. Ultimately, this contributes to societal well-being.

1.1 NEED FOR AN EXPENSE TRACKER APPLICATION ON AN IPHONE (SMARTPHONE)

A smartphone is built on a mobile operating system with advanced technology and computing capabilities. It all started with Symbian in 2000, which was the first modern mobile OS on a smartphone followed by Blackberry in 2002, Apple iPhone in 2007, Android

in 2008 etc. Global smartphone sales increased 47.3 % to 149 million units in the fourth quarter of 2011, according to market research firm Gartner [1].

According to the smartphone recent sales stats globally, IDC says that over 153 million smartphones sold worldwide in Q2 2012 [2].

Smartphone shipments based on Mobile OS as per Q2 2012 by IDC in Figure 1.1 [2].

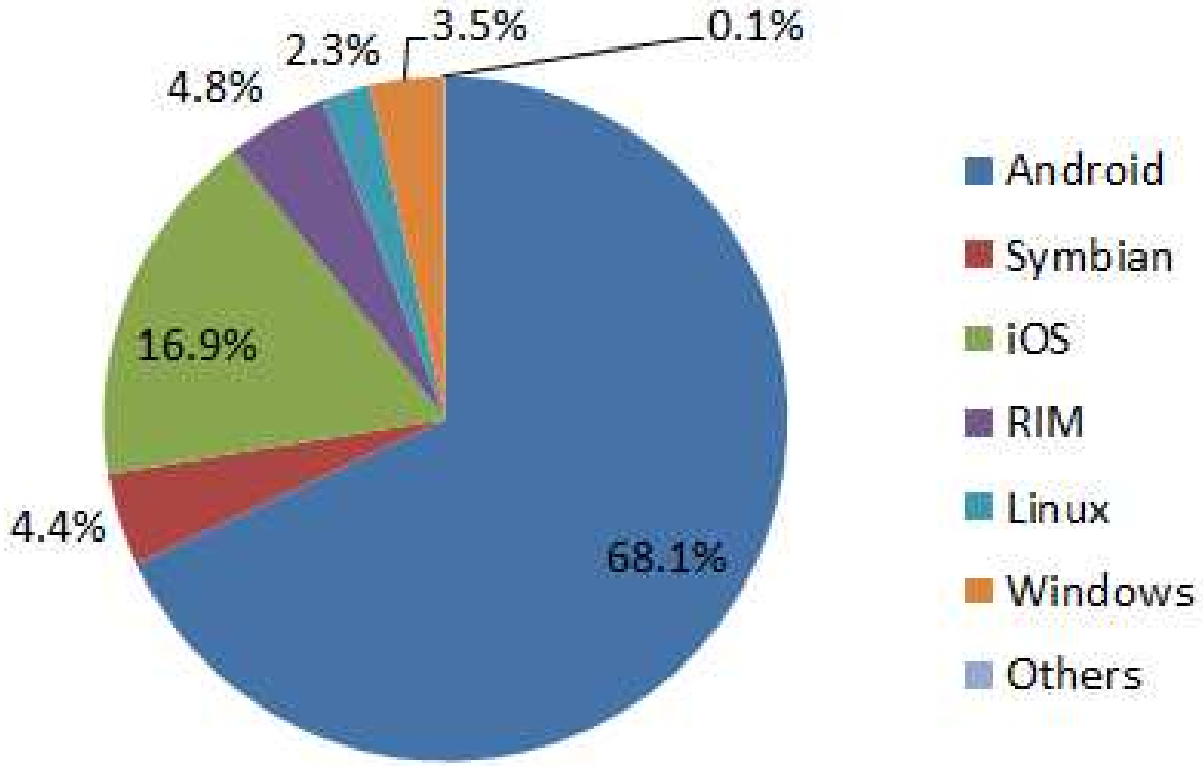

Figure 1.1. Share of worldwide 2012 Q2 smartphone sales by OS, according to IDC. Source: M. BROWNLOW. *Smartphone statistics and market share*. Email-Marketing-Reports, http://www.email-marketing-reports.com/wireless-mobile/smartphone-statistics.htm, accessed October 2012, October 2012.

The above pictorial representation shows that a lot of people are using Apple's iPhone device which works on iOS. With the launch of the new iPhone5 recently whose sales hit 5 million units in first weekend [3] clearly indicates that many people are using it now and thus arises a need for an application that serves to manage their lifestyle by managing their finances accordingly. The sole reason behind the need for this application is that when you start finding out where you are spending your money, it becomes easy to figure out how you can save money by cutting back or even eliminating spending in some areas. You can concentrate on saving even a little bit on your biggest expense categories, compared to

cutting back a lot on smaller expenses. As this tracking expense becomes a habit, you can get a good picture of how much money you need to maintain your lifestyle.

Budget Setting and expense tracking both alter consumer choice. Mental Budgeting leads people to overconsume some goods and underconsume others. Because budgets are set before consumption opportunities arise, they sometimes overestimate or underestimate the money required for a particular amount. On the other hand, expense tracking shows that some expenses are more likely to produce over-or-underconsumption. As people track their expenses, expenses that are relatively easy to categorize – those that are more typical examples of their categories –will be the most subject to the rigors of budgeting [4, 5].

A number of students at SDSU were asked how much they spend per month on their needs and requirements, and nobody was able to provide an exact answer. Thus, they do not know where their money is going. They liked the idea of such an expense tracking tool where with just a few inputs everyday, you can make yourself organized and make your life a bit easier in the long run.

There is a web application and an iPhone app such as "Mint" [6], which handles finances, but users need to input the bank information or the credit card information that they will use to make expenses so that Mint can function. Although the system is SSL encrypted, entering the credit card information is somewhat that any user will hesitate to do at first. If an user possesses many credit and debit cards from which s/he can make expenses, then s/he has to enter them all in Mint in order for it to function and manage the finances. There is no provision to manage cash track activity in Mint. In XpensTrak, I made it pretty clear for the user to choose his payment source either by cash, credit card, debit card, check or a gift card. Please refer to the Appendix for the comparison between Mint and XpensTrak.

1.2 iPHONE OPERATING SYSTEM TECHNOLOGIES

Apple's iPhone, iPod Touches and iPad devices run on iOS which is the operating system. The iOS SDK comprises of the interfaces, tools and components required to develop and deploy native applications. Web applications are created by the use of HTML, CSS and javascript code. The development environment to build the iOS applications is Xcode where we can create, test, debug our apps. The apps are written in Objective C Programming language. We can run the apps on iOS Simulator or on an attached iOS Device. For that, we

need to get an Apple Developer account if we want to test the app on a real device. Everything works in terms of packages called Frameworks in iOS. Basically, a framework is a directory containing the libraries and the necessary resources like images, header files that we need to import into the project. The higher-level frameworks should be preferred over lower-level frameworks as they provide much more functionality and features like sockets and threading with a good reduction in the length of the code.

The implementation of iOS technologies presented as set of layers as shown in Figure 1.2 [7].

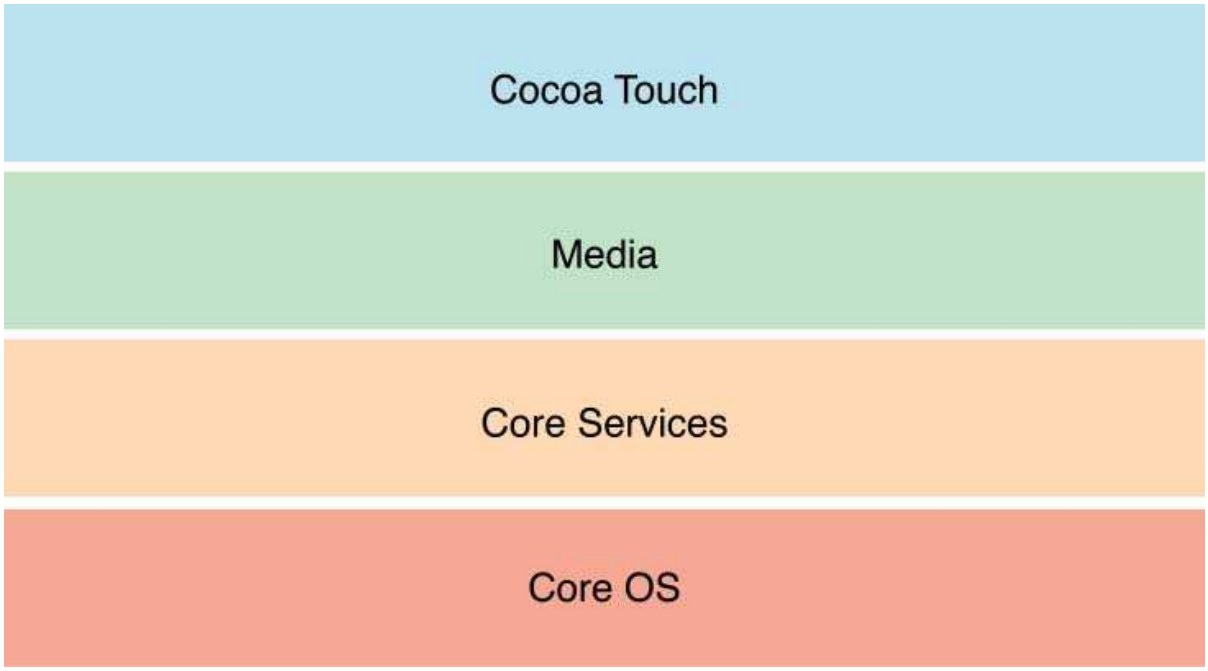

Figure 1.2. Layers of iOS. Source: APPLE, *iOS technology overview*. Apple, http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html, accessed October 2012, n.d.

The Cocoa Touch layer contains the most important frameworks required to develop an application. With the launch of iOS6 this fall 2012, features such as Auto Layout, UI State Preservation have been added. Auto layout allows the user to define set of rules to lay out the elements in user interface design. It's a very intuitive approach than "Springs and struts" used in earlier iOS versions ad it provides more flexibility and better layering between view objects. Another feature called StoryBoards was added in iOS 5 which shows the application flow in one interface file itself rather than making different user interface files for each view.

The frameworks used in Cocoa Touch layer comprises of Address Book, Event Kit UI, Game Kit, MapKit, iAD, Twitter, UIKit frameworks. UIKit framework is a key framework used by iOS to implement all the core basic features in an app such as Application Management, User Interface Management, Graphics, Multitasking and a lot more related to device specific features too.

The Media layer is very important in order to incorporate graphics, audio and video technologies in an iOS application. The Media layer comprises of Assets Library, AVFoundation, Core Audio, Core Graphics, Core Text, Core Video, Media Player, QuartzCore and many other frameworks.

The Core Services Layer provides the high level features such as iCloud Storage which allows the application to write the data to a central location in a cloud and access them from a computer or any iOS device. The biggest advantage is even if the user loses the device, the information on that device is never lost if it's on iCloud. Another features includes SQLite, XML Support, Automatic Reference counting that eliminates the need for manual memory management and Block supports. It consists of Ad Support, CFNetwork, Core Data, Accounts, Address Book, Core Foundation, Core Media, Foundation, Mobile Core Service and other frameworks.

The Core OS Layer comprises of the low level features mostly implemented in C and provide support for Threading, Networking, File System access, memory allocations and math computations. These are basically used by other frameworks or technologies they are built upon.

1.3 CORE APPLICATION OBJECTS

The UIKit Framework provides the components and functionality from the time the app starts and till it exits. The core application objects used in iOS [8] are:

1. UIApplication object : This object manages the application event loop, receives events from the system and works in sync with the code in app delegate object.
2. App Delegate object : This object is responsible for app launch time and manage transitions to and from the background.
3. Documents and data model objects : Data model objects store specific information to the app and document objects which is a subclass of UIDocument performs the same function of storing info but is flexible and convenient to group bunch of data that belongs to a single file.

4. View Controller objects : This object is responsible for presenting the application on the screen and manages the views and subviews within it.
5. UIWindow object: Every app has one window which is responsible for showing the views in it. This object serves to host views.
6. View, Control and layer objects: These objects primarily represent the visuals on the application. The controls are objects such as buttons, switches, labels, text fields etc. and views are objects that's shows the content on the screen in a rectangular area. Core Animation layers are data objects that provide more custom drawings and content on the screen.

   The key objects in an iOS Application are shown in Figure 1.3.

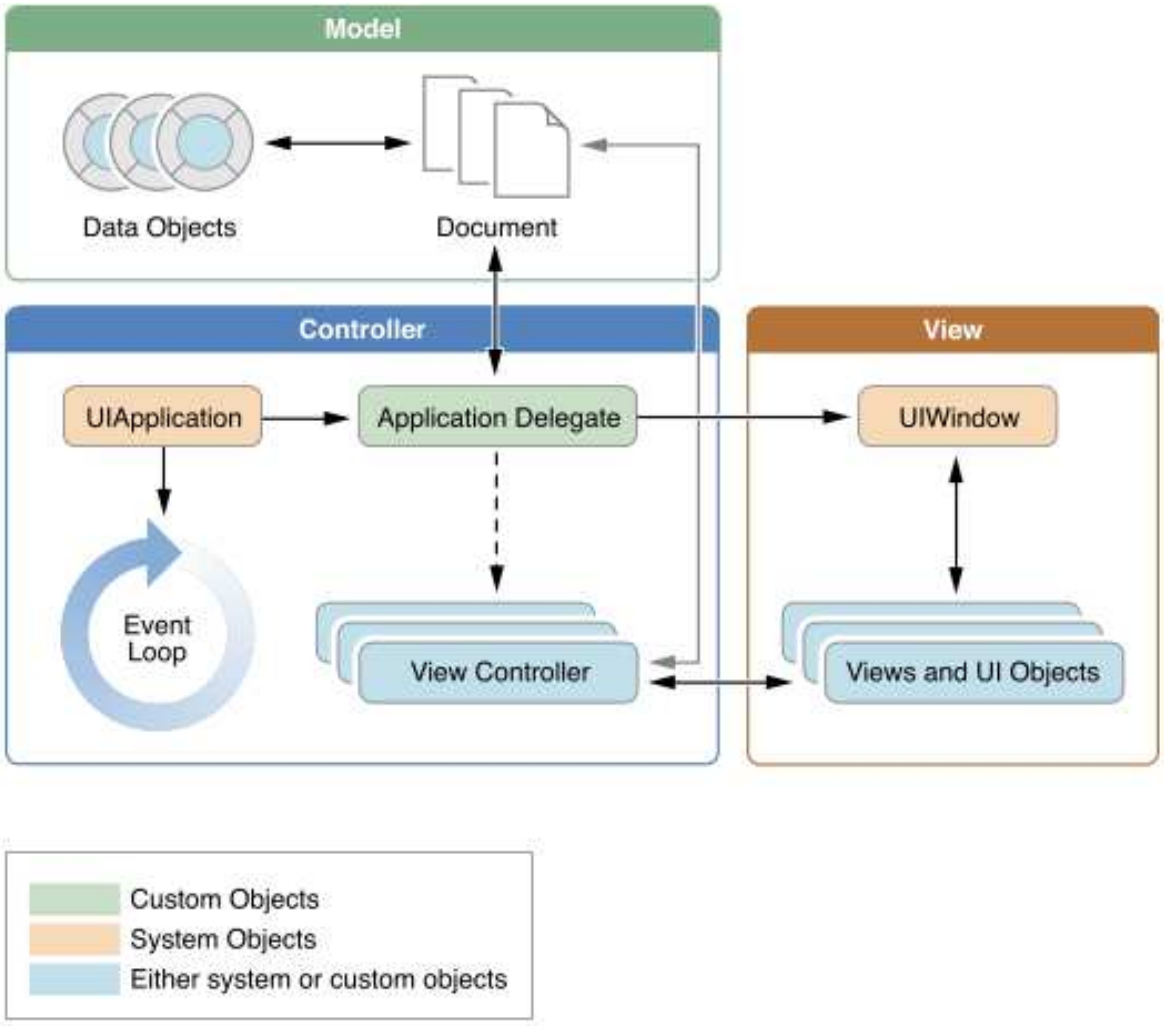

Figure 1.3. Key objects in iOS app. Source: APPLE, *Key objects of an iOS app*. Apple, http://developer.apple.com/library/IOs/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/AppArchitecture/AppArchitecture.html#//apple_ref/doc/uid/TP40007072-CH3-SW2, accessed October 2012, n.d.

1.4 Application States

An iOS application can be running in the background or foreground and it becomes very important to know when there is a transition in the state. The behavior of an app changes when it is in background state as the operating system puts limitations to what the app can perform in order to improve the battery life of the phone as well as manage it properly with the foreground app. The system uses notifications while making the transitions like when changing the state from background to foreground or foreground to background. While in the suspended state, no notifications are sent by the operating system.

An iOS application is usually in one of the following states:

1. Not Running: The application has not been launched in this state or being terminated by the system while running.
2. Inactive: The application is not receiving any events but still running in foreground.
3. Active: The application is running in foreground and receiving events as well.
4. Background: The application in this state is in background and still working as in the code is executing behind the scenes and the app is not killed. This state usually comes when the application is on it's way to being suspended or when the application needs some extra execution time.
5. Suspended: The application in this state is in background but not executing any code meaning the application still remains in memory but it's not working in terms of executing the code. The operating system does not notify the app when it moves it to thes suspended state.

Figure 1.4 [9] shows the various state changes in an iOS application.

1.5 Application Launch Cycle

Figure 1.5 [9] shows the flow when the application is launched in the foreground. When the user taps the app icon, it calls the main method which in turn calls the UIApplicationMain( ) method. The main UI file is loaded and the application is first initialized by the app delegate applicationWillFinishLaunchingWithOptions:method and applicationDidFinishLaunchingWithOptions: method at final initialization. When the application is running and active, applicationDidBecomeActive: method is called from the app delegate. A series of events are executed which are handled by the code itself. When the

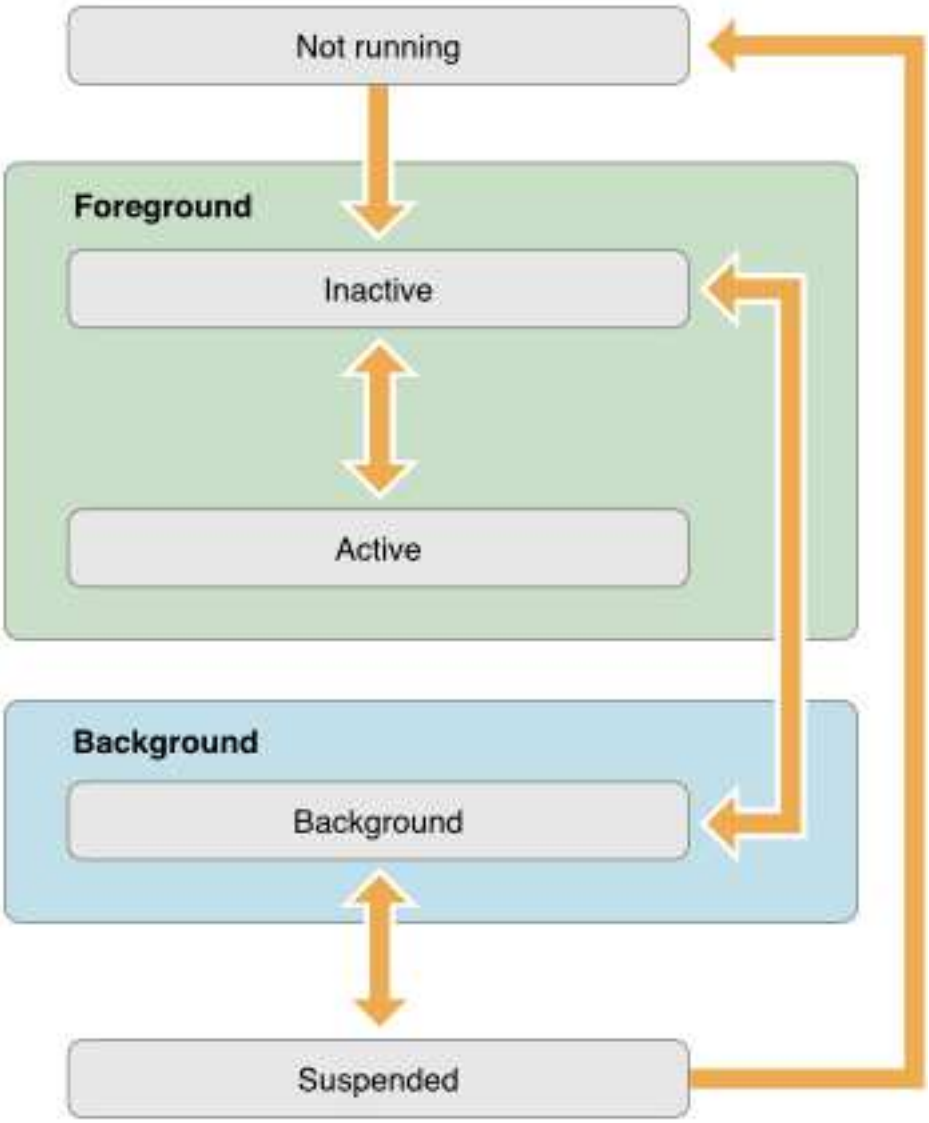

Figure 1.4. State changes in iOS app. Source: APPLE, *App states and multitasking*. Apple, http://developer.apple.com/library/IOs/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html#//apple_ref/doc/uid/TP40007072-CH4-SW1, accessed October 2012, n.d.

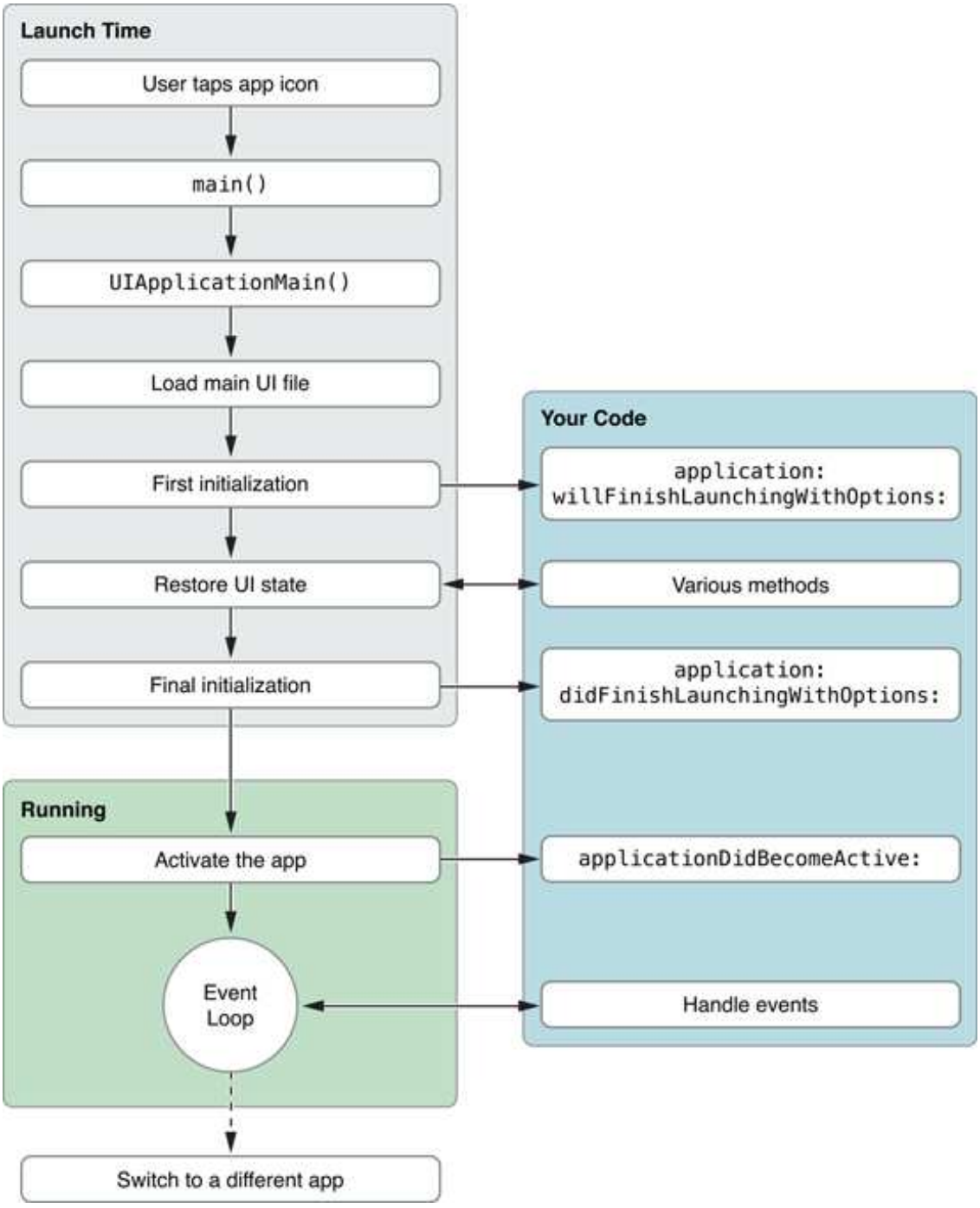

Figure 1.5. The app launch cycle. Source: APPLE, *App states and multitasking*. Apple, http://developer.apple.com/library/IOs/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html#//apple_ref/doc/uid/TP40007072-CH4-SW1, accessed October 2012, n.d.

app quits foreground, applicatonDidResignActive: method is called and when app enters the background, applicationDidEnterBackground: method is called.

When alert-based interruptions occur such as an incoming phone call while the app is running, then the app moves to inactive state for that moment calling the applicationWillResignActive: method and moves back to the active or the background state when the user dismisses the alert calling the applicationDidBecomeActive: method.

CHAPTER 2

IMPLEMENTATION OF THE MOBILE APPLICATION

2.1 DESIGN AND DEVELOPMENT

Prior to building any application from scratch, it is important to plan the design flow of the application. The user interface design is pretty essential to know how the application is intended to work. Planning an application definitely involves some specific strategies.

Many strategic questions come to the mind while designing and planning an application like there might be apps in the app store similar to your idea, what the need of such an application in the market is, what features make your app unique, what will make the people use your app and not the ones designed by others, how much time the application will take to develop, what user requirements either small or big should be considered, what category should the app fall into, etc.

After planning the application, next comes the sketching or wire-framing it. Wireframing is simply designing the flow of the application, describing how the navigation of the views will work. OmniGraffle [10] is a very flexible tool for wire-framing an application. If the app is designed involving a team, it becomes very essential to use such kind of tool to help design the app and distribute the idea among the team. After the proper flow of the app, next comes the development phase.

Apple provides a number of tools to help developers build applications for iOS and Mac.

Xcode is an IDE that is available from the Mac App Store and the developers can sign in and download it to start building applications. Xcode comes with the Simulators on which we can run and test the app, Interface builders to build the interfaces visually inside the app, command line tools required to develop, cocoa frameworks and various testing tools such as Instruments to detect allocations and memory leaks in the application. The programming language used to write the application in Xcode is Objective-C which is a superset of C.

Xcode 4.5 is used to develop this mobile application. Figure 2.1 shows the Xcode IDE.

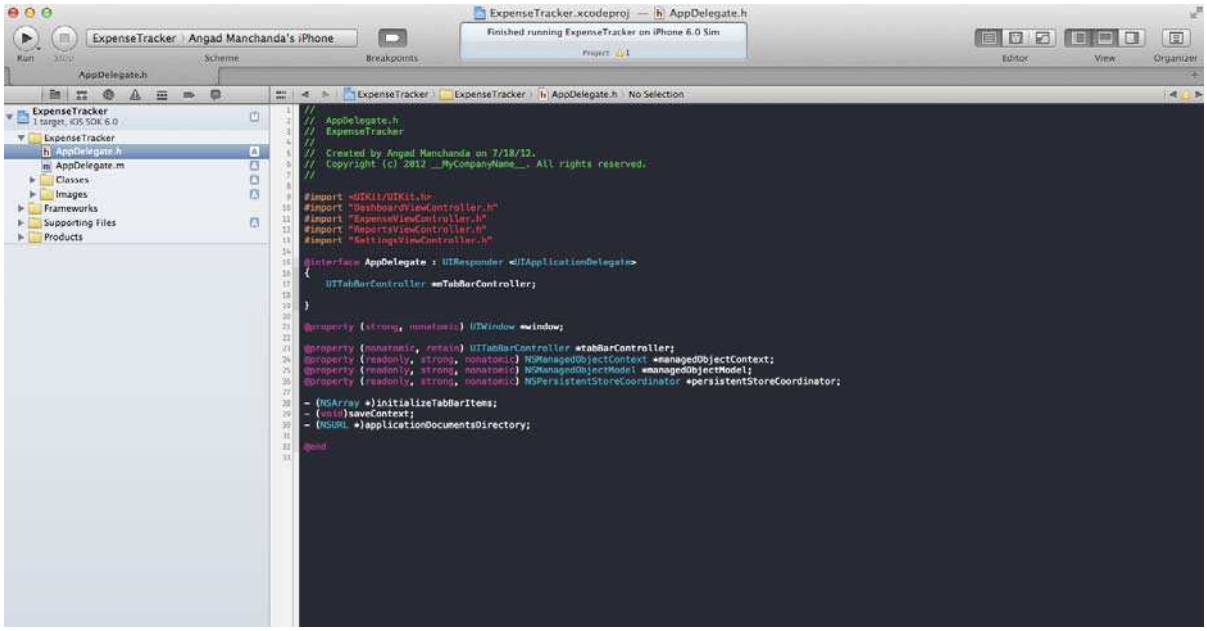

Figure 2.1. XpenseTrak iOS app on Xcode IDE.

2.2 TAB BAR VIEW

When the application is first launched, the applicationDidFinishLaunching: method is called which resides in the AppDelegate class. The AppDelegate class consists of the methods that are responsible for the launching, active, inactive, suspended, termination states of an application.

I designed a Tab Bar programmatically inside the app delegate which holds mainly the four Tab Bar items in this case (Dashboard, Money, Reports, Settings). Each Tab Bar Item has it's image to represent the items. The Tab Bar Controller acts as the Root View Controller and the tab bar items are presented with their respective navigation controllers [11]. The Tab Bar Controller has it's own delegate methods which recognizes which tab bar item is pressed and then it behaves accordingly. For example, Tab Bar Item 1 is touched, the tab bar controller delegate method will be called and will perform the required responsibilities. Figure 2.2 [12] shows the anatomy of a UITabBarController.

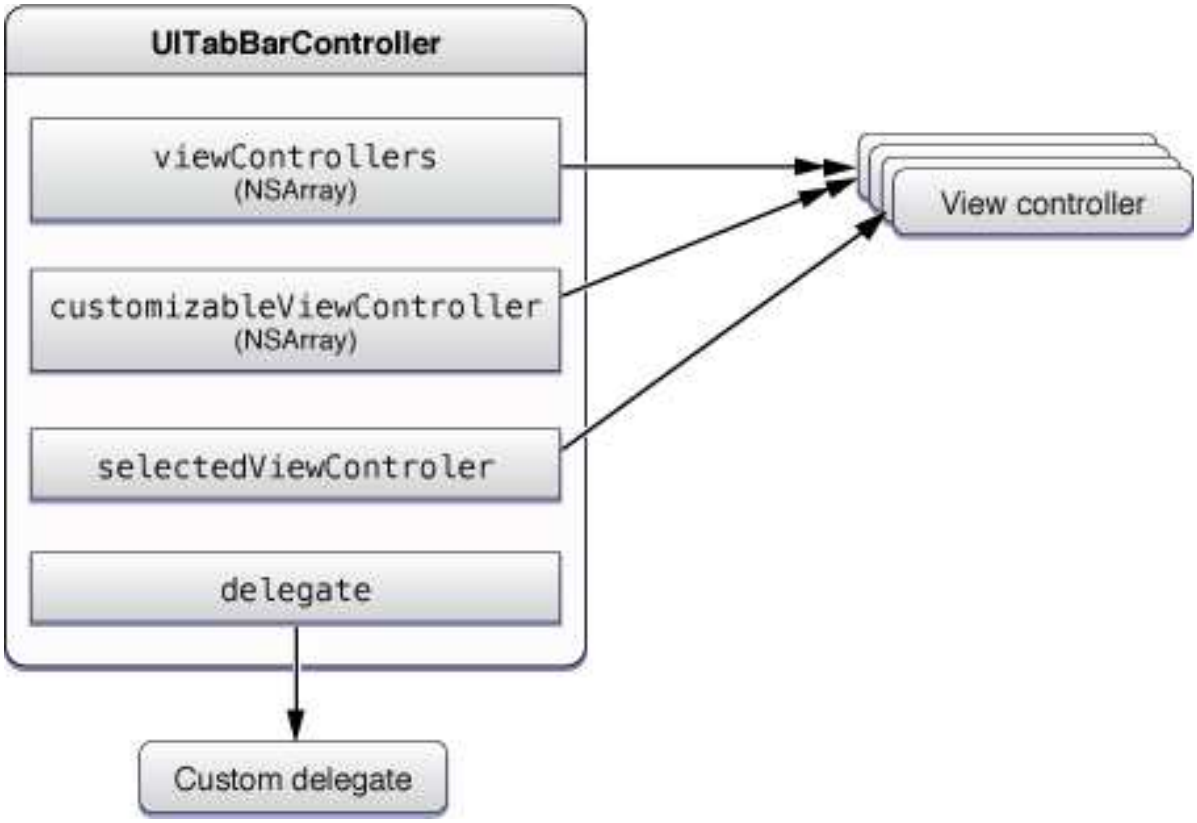

Figure 2.2. Tab bar controllers. Source: APPLE, *Tab bar controllers*. Apple, http://developer.apple.com/library/ios/#documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/TabBarControllers.html, accessed October 2012, n.d.

2.3 CORE DATA FRAMEWORK

Core Data framework provides an object graph management of the data model including persistence of the data [13]. Although Core Data is not a relational database, it uses SQLite as one of it's persistent stores. It provides the management of saving objects to and retrieving them from the storage. There is an object, known as managed object context which provides the functionality to Core Data framework. There is the Persistent Store Coordinator that acts as intermediary between the managed object contexts and the external data stores [14].

We create a schema in Core data with application entities, it's properties and relationships. The schema is represented by a managed object model, instance of NSManagedObjectModel.

The managed object model comprises of various entities and attributes and relationship among the different entities. Each entity has a name and is represented in the

code by an NSManagedObject subclass. Entity inheritance is a concept that is used in this application design. The entities that have common attributes in them can subclass from a parent entity. To retrieve the data using managed object context, a fetch request is created. It's basically selecting what data the user wants. A Predicate can be specified that the object should match some particular condition and an array of sort descriptors in which the objects should appear whether ascending or descending.

In this application design, various entities have been modeled and are related to each other through one to many or one to one relationships. The entities can have a to-many relationship or one-one relationship where each entity has a unique relationship name to the other entity.

Figure 2.3 shows the core data model used in the application.

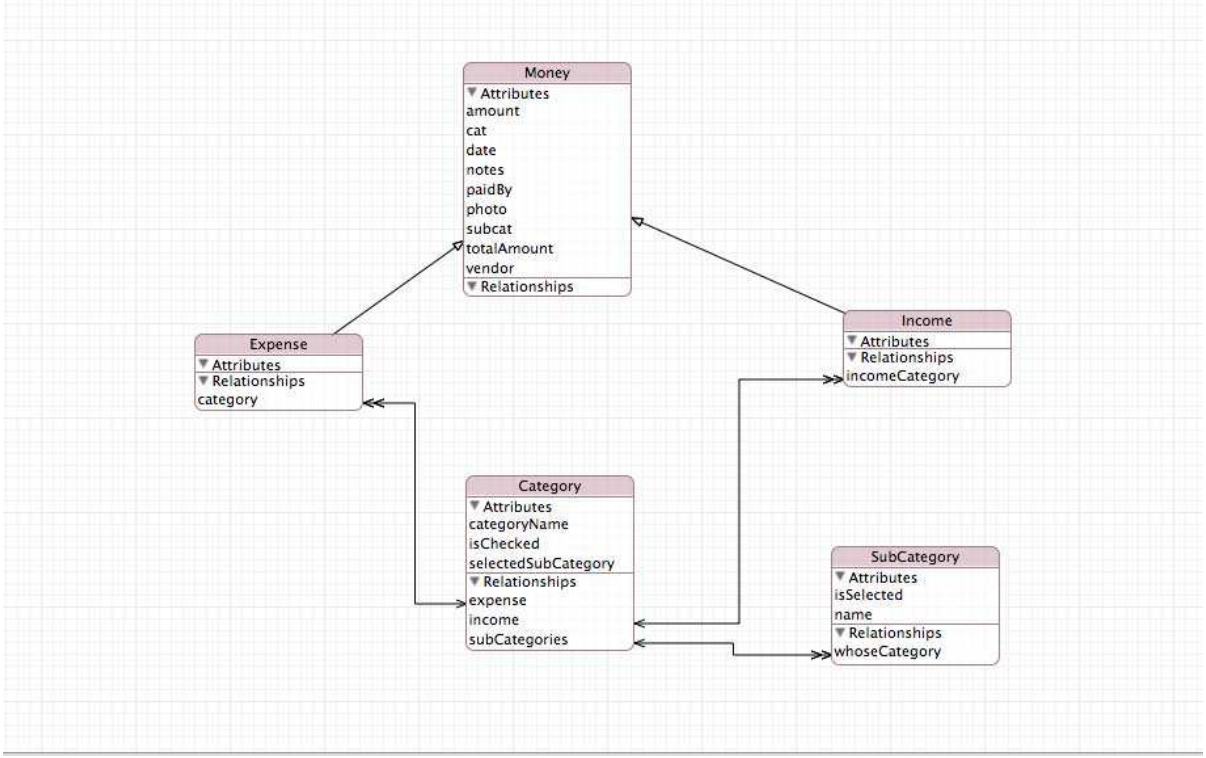

Figure 2.3. Core data model.

As Figure 2.3 shows, there are 5 entities designed named Money, Expense, Income, Category and SubCategory. Money is the parent entity with the attributes such as amount, category, date, notes, payment source method, photo, subcategory, total amount and the merchant. The reason the other two entities Income and Expense inherit from the parent

entity Money is that they are suppose to have the same attributes in them, so the common attributes are listed in Money entity. Then there is an entity called category with attributes such as categoryName, isChecked (to determine which is the selected category), selectedSubCategory (sub category selected for the particular category). Category has a to-many relationship with Expense and Income entities as there can be many expenses or incomes for a particular category. Then the category has to-many relationship with entity SubCategory as a category can have many sub-categories.

The results that are returned from the Core Data fetch request are managed by a fetched result controller that provide the data for the tableview used in the views. An instance of NSFetchedResultsController is created which acts as an instance variable of a table view controller.

```
- (NSFetchedResultsController *)fetchedResultsController
{
    if(_fetchedResultsController != nil)
    {
        return _fetchedResultsController;
    }

    AppDelegate * applicationDelegate = (AppDelegate *) [[UIApplication sharedApplication]
delegate];
    NSManagedObjectContext * context = [applicationDelegate managedObjectContext];

    NSFetchRequest * request = [[NSFetchRequest alloc]init];

    [request setEntity:[NSEntityDescription entityForName:@"Money"
inManagedObjectContext:context]];

    NSSortDescriptor *sortDescriptor1 =
    [[NSSortDescriptor alloc] initWithKey:@"date"
                                ascending:YES];

    NSArray * descriptors = [NSArray arrayWithObjects:sortDescriptor1, nil];

    [request setSortDescriptors: descriptors];
    [request setResultType: NSManagedObjectResultType];
    [request setIncludesSubentities:YES];

    [sortDescriptor1 release];

    self.fetchedResultsController = [[NSFetchedResultsController alloc]
initWithFetchRequest:request
                                                    managedObjectContext:context
                                                      sectionNameKeyPath:nil
```

```
                                                    cacheName:nil];
    self.fetchedResultsController.delegate = self;

    [request release];

    NSError *anyError = nil;

    if(![_fetchedResultsController performFetch:&anyError])
    {
        NSLog(@"error fetching:%@", anyError);
    }

    return _fetchedResultsController;

}
```

When a fetched result controller is initialized, we provide four parameters:

1. Fetch request that contains at least one sort descriptor for the resulting order of the objects.
2. Managed object context which the fetched results controller uses to execute the fetch request.
3. Section name key path that is optional which would order the results by section. Making it nil would just make one section and the results would show in that section itself.
4. Cache name used, which is optional. Making it nil would prevent caching.

To execute the fetch request, performfetch: method is called on the controller. The class that uses the fetched results controller implements the NSFetchedResultsController delegate. The controller basically notifies the delegate when the objects changes locations or when there is a modification of the sections.

The setIncludesSubEntities will produce the results for both the entities Expense and Income as the request is made on the parent entity Money.

The NSFetchedResultsController delegate methods are:

```
- (void)controllerWillChangeContent:(NSFetchedResultsController *)controller

- (void)controller:(NSFetchedResultsController *)controller didChangeSection:(id
<NSFetchedResultsSectionInfo>)sectionInfo atIndex:(NSUInteger)sectionIndex
forChangeType:(NSFetchedResultsChangeType)type

- (void)controller:(NSFetchedResultsController *)controller didChangeObject:(id)anObject
atIndexPath:(NSIndexPath *)indexPath
forChangeType:(NSFetchedResultsChangeType)type newIndexPath:(NSIndexPath
*)newIndexPath
```

```
- (void)controllerDidChangeContent:(NSFetchedResultsController *)controller
```

2.4 EXPENSE/INCOME VIEW

The expense/income view will contain the list of expenses/incomes made for a particular date. The user can either enter an expense or an income depending on what he chooses from the Block Action Sheet which is explained below.

2.4.1 Block Action Sheet

After the Tab Bar is set up, the views for the Expenses and Incomes are set up. The "Money" tab bar item is pressed and it sets up a custom BLOCK ACTION SHEET [15] that allows the user to either enter the expenses or income or press cancel and make the alert go away. iOS SDK provides a built-in UIActionSheet component which is an action sheet view with buttons that performs the required actions. In this case, custom block action sheet is used because it has certain animation effects, the buttons are designed very well and gives the user interface a better look than the built in Alert View. Also, Blocks is a feature of Objective C 2.0 which allows us to write blocks of code and pass them along as objects. The custom Block Alert view uses Blocks in it's implementation and thus makes it very flexible, organized and readable in terms of the code. In case of the iOS built in action sheet, the code to initialize action sheet is written somewhere and it's button press actions occur somewhere else in the code, making it quite unreadable whereas the block action sheet is instantiated in the code and it's button press action occurs in the same block, making it organized and readable. Figure 2.4 shows the block action sheet view when the "money" tab bar item is pressed.

```
- (BOOL)tabBarController:(UITabBarController *)tabBarController
shouldSelectViewController:(UIViewController *)viewController
{
   if([viewController.title isEqualToString:@"Money"])
   {
      BlockActionSheet* sheet = [[[BlockActionSheet alloc]
initWithTitle:NSLocalizedString(@"Enter Expense or Income", nil)] autorelease];

      [sheet setDestructiveButtonWithTitle:NSLocalizedString(@"Expense", nil) block:^{

         moneyDetailController = [[MoneyDetailViewController
alloc]initWithNibName:@"MoneyDetailViewController" bundle:nil];
```

Figure 2.4. Block action sheet.

```
        moneyDetailController.expense = _expenseObject;
        moneyDetailController.pushedBtnValue = 1;

        UINavigationController *navControl = [[UINavigationController alloc]
initWithRootViewController:moneyDetailController];
        [self presentModalViewController:navControl animated:YES];
        [moneyDetailController release];
        [navControl release];
```

```
        }];

        [sheet addCustomButtonWithTitle:NSLocalizedString(@"Income", nil) block:^{

            moneyDetailController = [[MoneyDetailViewController
alloc]initWithNibName:@"MoneyDetailViewController" bundle:nil];
            moneyDetailController.income = _incomeObject;
            moneyDetailController.pushedBtnValue = 2;


            UINavigationController *navControl = [[UINavigationController alloc]
initWithRootViewController:moneyDetailController];
            [self presentModalViewController:navControl animated:YES];
            [moneyDetailController release];
            [navControl release];

        }];

        [sheet setCancelButtonWithTitle:NSLocalizedString(@"Cancel", nil) block: ^{}];

        [sheet showInView:[self view]];

    }
    return YES;
}
```

The block action sheet is allocated and initialized with a title in the tab bar controller delegate method. When the Money tab item is pressed on the tab bar controller, it pops up the action sheet in the view. The buttons (Expense and Income) are set up and their corresponding actions take place in their respective block itself pushing to the new view which will allow user to enter expenses or income in detail.

2.4.2 Calendar View

To track down the expenses /income made, it's very important to know what day the expense was made. For that, an existing custom UI Component called PMCalendar [16] is used. These components are built upon the various built in iOS frameworks such as Core Graphics, Core Text frameworks. It does not require any third party framework but all these custom components are available open source so that that they can be included in the project and then can be customized enough to make it work accordingly in the application. The calendar shows the current date being selected and will update the date automatically

everyday and the user can basically select whatever date and see the expenses made that day keeping it very flexible and organized.

On the calendar button pressed shown in Figure 2.5, the calendar view pops up and the user can see or select the dates on which the expenses were made. All the interface components (buttons, text fields, labels) are build in the Interface Builder provided by Xcode, they are referenced by IBOutlets and their corresponding actions are reperesented by IBActions.

Figure 2.5. PMCalendar view.

```
- (IBAction)showCalendar:(id)sender
{
    self.pmCC = [[PMCalendarController alloc] initWithThemeName:@"default"];
    pmCC.delegate = self;
    pmCC.mondayFirstDayOfWeek = NO;

    [pmCC presentCalendarFromView:_showCalendarBtn.superview
        permittedArrowDirections:PMCalendarArrowDirectionDown
                        animated:YES];

    [self calendarController:pmCC didChangePeriod:pmCC.period];

    [pmCC release];

}
```

A PMCalendar object is instantiated when the button is presented and the view is being presented and it's delegate method didChangePeriod: is called when the user selects the dates on the calendar view and it updates to that selected date and shows the expense record.

2.4.3 Expense/Income Detail View

When the expense or the income button on the Block Action Sheet is pressed, it navigates to the next view called the MoneyDetailViewController which basically contains some fields that are supposed to be entered by the user in order to save their daily expenses or incomes.

User can take the photo of the receipt (bills), enter the amount, merchant and category. Date will be automatically set to the current date and user has the option to fill sub-category, payment source and some notes about the expense/income. The view is presented in a table view style called UITableView in iOS which presents the data in rows. When Category row is pressed, it takes the user to another view called CategoryView with some pre-filled entries in the database. Categories can be added, edited and deleted as well depending on the user needs and requirements. Sub-Category view is displayed when the user taps a category and can add a sub-category related to the category being tapped. The payment source Row presents another screen with a tableview displaying various sources such as Cash, Credit card, Debit Card, Gift Card and Check. Figure 2.6 shows the expense detail view.

Figure 2.6. Expense detail view.

Expense detail view consists of the above fields when entered by the user. Figure 2.7 shows the block action sheet when the photo button is clicked by the user in order to take the picture of the receipt.

The Take Photo button will open up the camera, Choose from photo library button will take the user to the iPhone photo gallery and choosing any picture will save it to the photo placeholder. The cancel action will just hide the action sheet. The

Figure 2.7. Photo button action.

UIImagePickerController class is responsible for taking pictures and for choosing saved images from the photo gallery and use them in the application. A source type is assigned to UIImagePickerController for different actions. For taking pictures from camera, source type of UIImagePickerControllerSourceTypeCamera is used and for choosing saved images from the gallery, UIImagePickerControllerSourceTypePhotoLibrary is used. The class that implements this controller also conforms to the UIImagePickerControllerDelegate protocol.

```
imagePicker=[[UIImagePickerController alloc] init];
imagePicker.sourceType=UIImagePickerControllerSourceTypeCamera;
imagePicker.delegate=self;
imagePicker.allowsEditing=YES;
[self.navigationController presentModalViewController:imagePicker animated:YES];
[imagePicker release];
```

The delegate method used with UIImagePickerController is notified when a new image is taken or an image is choosen from the photo gallery.

```
-(void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{

   if(imagePicker.sourceType == UIImagePickerControllerSourceTypeCamera)
   {
      [receiptBtn setTitle:@"" forState:UIControlStateNormal];
      _templateImgView.templateImage.image=[info
objectForKey:@"UIImagePickerControllerEditedImage"];
      UIImageWriteToSavedPhotosAlbum(_templateImgView.templateImage.image, nil,
@selector(image:didFinishSavingWithError:contextInfo:), nil);

   }

   else if(imagePicker.sourceType == UIImagePickerControllerSourceTypePhotoLibrary)
   {
      [receiptBtn setTitle:@"" forState:UIControlStateNormal];

      _templateImgView.templateImage.image=[info
objectForKey:@"UIImagePickerControllerEditedImage"];
   }

   [picker dismissModalViewControllerAnimated:YES];

}
```

Figure 2.8 shows the Income Detail view quite similar to the expense detail view.

Income detail view consists of the same fields as the expense detail view and looks like above when filled by the user. When the save button is pressed, the income is saved for that date which is displayed in another view.

iOS includes a UITableView component in UIKit Framework which consists of showing the data in a tableview which can be vertically scrolled. The tableview styles can be plain or grouped and consists of zero or more number of sections and rows that are made of UITableViewCell objects. UITableView uses the tableview cells objects to draw the rows. The UITableView has an object that acts as the data source and an object that acts as a a

Figure 2.8. Income detail view.

delegate. It can be application delegate or a custom UITableViewController object. It must correspond to the tableview data source and table view delegate protocols. The data source and delegate methods for UITableView are:

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section

```
- (UITableViewCell *)tableView:(UITableView *)tblView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
    *)indexPath
```

2.4.4 Category/Sub-Category View

When the category row on the Expense/Income Detail view is pressed, it brings to another tableview with some pre-filled entries in the database. The user can press the edit button to delete the existing categories or add new categories to the existing list. The Subcategory view is displayed in a similar fashion when a particular category is being tapped. The user has the option to add a sub-category to the related category. If all the rows are entered, the user has the flexibility to see the results (expenses) in the reports tab based on Category, Sub-Category, vendor, payment source etc. Figure 2.9 shows the category view. When any of the views is being launched, viewDidload method in iOS is being called which instantiates the loading of the view. There are methods like viewWillAppear, viewDidAppear with little differences and are called accordingly depending on the actions the view has to perform.

These pre-filled entries are stored in Core Data database model. In order to persist the category/expense/income data, Core Data has been used as described above in Section 2.2.

```
- (void)insertCategory:(NSString *)category
{
    AppDelegate * applicationDelegate = (AppDelegate *) [[UIApplication sharedApplication]
delegate];

    NSManagedObjectContext * context = [applicationDelegate managedObjectContext];

    _category = [NSEntityDescription insertNewObjectForEntityForName:@"Category"
                                          inManagedObjectContext:context];

    _category.categoryName = category;

    NSError * error = nil;
    [context save:&error];
}

- (void)importCoreDataDefaultCategories
{
    [self insertCategory:@"Auto"];
    [self insertCategory:@"Income"];
    [self insertCategory:@"Entertainment"];
```

Figure 2.9. Category view.

```
[self insertCategory:@"Food"];
[self insertCategory:@"Home"];
[self insertCategory:@"HouseHold"];
[self insertCategory:@"Personal"];
[self insertCategory:@"Electronics"];
[self insertCategory:@"Transportation"];
[self insertCategory:@"Travel"];
[self insertCategory:@"Utilities"];
[self insertCategory:@"Holidays"];
```

```
}
```

When the view is launched viewDidLoad: method is called and it checks in there if there are no fetched objects, it imports the above existing categories.

```
- (void)viewDidLoad
{
    if (![[self.fetchedResultsController fetchedObjects] count] > 0 ) {
        [self importCoreDataDefaultCategories];
    }
    else {
    }

    self.navigationItem.rightBarButtonItem=self.editButtonItem;

    [super viewDidLoad];
}
```

When a new category is added to the list, it is saved by the following save: method.

```
- (void)saveItem:(id)sender // adding a new category to the list.
{

    if([[_addTxtField text] length] == 0)
    {
        NSString* title = NSLocalizedString(@"Category Missing", nil);
        NSString* message = NSLocalizedString(@"Add a new Category", nil);

        BlockAlertView* alert = [BlockAlertView alertWithTitle:title message:message];
        [alert addButtonWithTitle:NSLocalizedString(@"OK", nil) block:^{ }];
        [alert show];
        return;
    }

    AppDelegate * applicationDelegate = (AppDelegate *) [[UIApplication sharedApplication]
delegate];

    NSManagedObjectContext * context = [applicationDelegate managedObjectContext];

    _addCategory = (Category*) [NSEntityDescription
insertNewObjectForEntityForName:@"Category" inManagedObjectContext:context];

    _addCategory.categoryName = _addTxtField.text;

    NSError * error = nil;
    [context save:&error];

 [self.navigationControlle dismissModalViewControllerAnimated:YES];

}
```

The new objects are added by referencing to the entity Category and then it's saved to the managed object context and can be retrieved from it later on by a fetch request if needed. New entries are added by using insertNewObjectForEntityForName: method on NSEntityDescription which is provided by NSManagedObjectModel class.

Block Alert view is used in order to pop up an alert when the user tries to save a new category/subcategory without adding a name to it. iOS provides UIAlertView component just like UIActionSheet described above, but Block Alert view is more flexible and got many advantages over the built in component in iOS as mentioned in 2.4.1.

Figure 2.10 and 2.11 shows the view for adding sub-category and the block alert while trying to add a subcategory without a name.

The sub-categories can be deleted or added by pressing the plus button on the left and that will pop up a view to add a new sub-category. The user can create and add as many categories/subcategories he wants to related to the expenses/incomes.

The custom built Block Alert view is popped up when the user tries to save the new sub-category without a name for it.

2.4.5 Date Picker View

When the date row on the Expense/Income Detail View (Figure 2.6 and 2.7) is tapped, it moves to a date picker view so that the user can select any date in the picker to see the expense/income made or probably add an expense/income for a particular date if he forgot to add it on that date itself (see Figure 2.12).

```
(void)addDatePicker
{
    _datePicker.datePickerMode = UIDatePickerModeDate;
    [_datePicker addTarget:self action:@selector(datePickerValueChanged:)
forControlEvents:UIControlEventValueChanged];

    NSDate *presentDate=[NSDate date];

    _datePicker.date = presentDate;
    [_datePicker setDate:presentDate animated:YES];

    _dateLbl.text = [NSString stringWithFormat:@"%@",[dateFormatter
stringFromDate:presentDate]];


    NSDate *d =[_datePicker date];
```

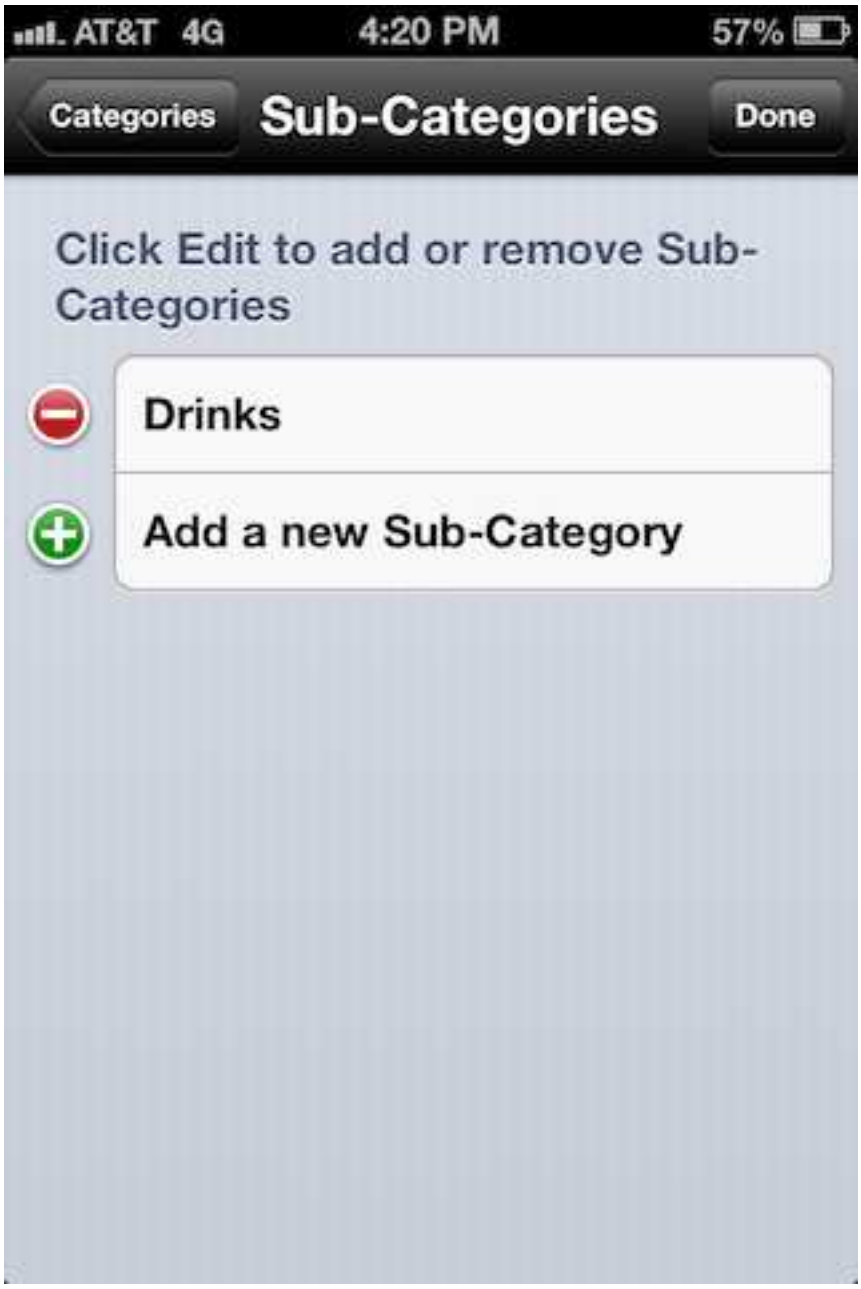

Figure 2.10. Sub category view.

```
unsigned units = NSYearCalendarUnit | NSMonthCalendarUnit | NSDayCalendarUnit;
NSCalendar *calendar = [[NSCalendar alloc]
                        initWithCalendarIdentifier:NSGregorianCalendar];

NSDateComponents *components = [calendar components:units fromDate:d];

NSInteger year = [components year];
NSInteger month = [components month];
NSInteger day=[components day];
```

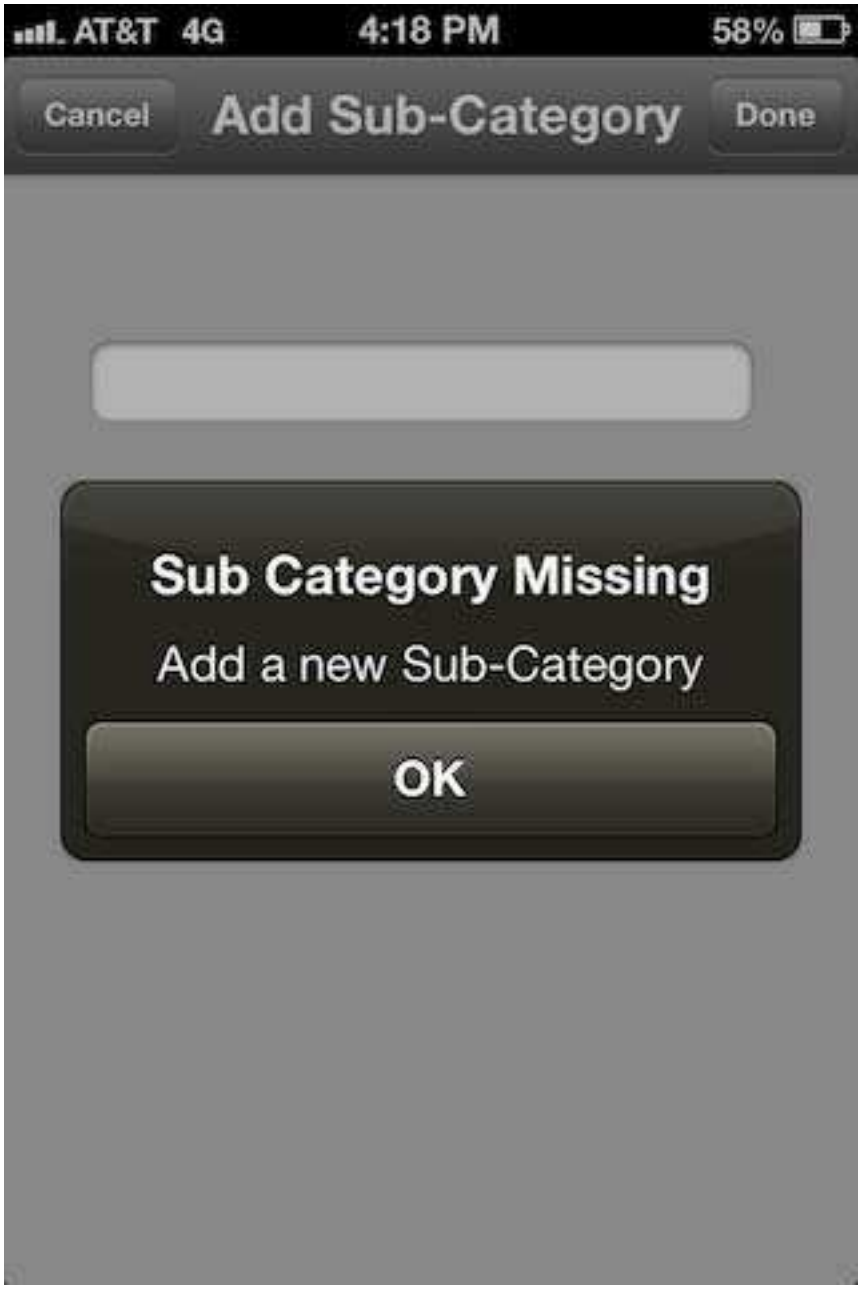

Figure 2.11. Block alert view.

```
    [calendar release];

    dateDayIndex=day;
    dateMonthIndex=month;
    dateYearIndex=year;

}
```

Figure 2.12. Date picker view.

The current date is set up on the date picker. It's a UIDatePicker component available in iOS built in interface. The UIDatePicker provides the date and date picker modes which is set up to show the month, date and year. When the date picker is rotated to change the date, datePickerValueChanged: method is called to set the date set up by the user.

2.4.6 Payment Source View

Whenever the expense is made, it's pretty important to know the payment source method being used. Expenses/Incomes are being tracked by the payment source method so that the user can know by what source he is spending or making money. The source can be cash, credit card, debit card etc. Figure 2.13 shows the payment sources used in the application.

2.5 DASHBOARD VIEW

The Dashboard View (see Figure 2.14) basically shows the summary of expenses/incomes based on Category, vendor and the payment source. The summary is shown weekly, monthly and yearly in a table view listing the category and total expense/income amount related to it so that the user can keep a detailed track where he's spending and where he needs to cut down the maximum in order to ensure a good living which is the sole purpose of this software tool.

2.6 REPORTS VIEW

The Reports View (see Figure 2.15), as the name says shows the reports or the results from the feed of an user's inputs for an expense/income generated. The user has the flexibility to see the total expense/income amount made from a certain period of time. The time duration can be chosen by the user and the reports can be based on a variety of choices such as category, vendor, subcategory, payment source etc.

2.7 SETTINGS VIEW

The application has a settings view which allow the users to choose whatever setting options they want to use in the application. The settings view has been implemented by using a third party open source library called InAppSettingsKit [17] for iOS. This library is an open source solution that provides in-app settings inside an iPhone application. A settings bundle is incorporated with a Root plist file that specify the UI elements with the NSUserDefault keys. The elements such as text fields, buttons, labels, toggle switches etc can be included in the pList with the key values that will be appear in the form of a tableview in settings application of iPhone as well as in the in-app settings of the application.

Figure 2.13. Payment source view.

Figure 2.16 shows the Root pList file in the settings bundle which forms the elements in the settings view.

The settings view sublasses IASKAppSettingsViewController which is a table view controller with the connection of the UI elements and the NSUserDefault keys. NSUserDefaults is a way to persist the data in iOS. The value of the keys are preserved.

Figure 2.17 shows the settings view in this application.

Figure 2.14. Dashboard view.

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
        return [self.settingsReader numberOfSections];
}
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {
    return [self.settingsReader numberOfRowsForSection:section];
}
```

Figure 2.15. Reports view.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
        IASKSpecifier *specifier  = [self.settingsReader specifierForIndexPath:indexPath];
        if ([specifier.type isEqualToString:kIASKCustomViewSpecifier] && [self.delegate
respondsToSelector:@selector(tableView:cellForSpecifier:)]) {
            UITableViewCell* cell = [self.delegate tableView:tableView cellForSpecifier:specifier];
            assert(nil != cell && "delegate must return a UITableViewCell for custom cell types");
            return cell;
```

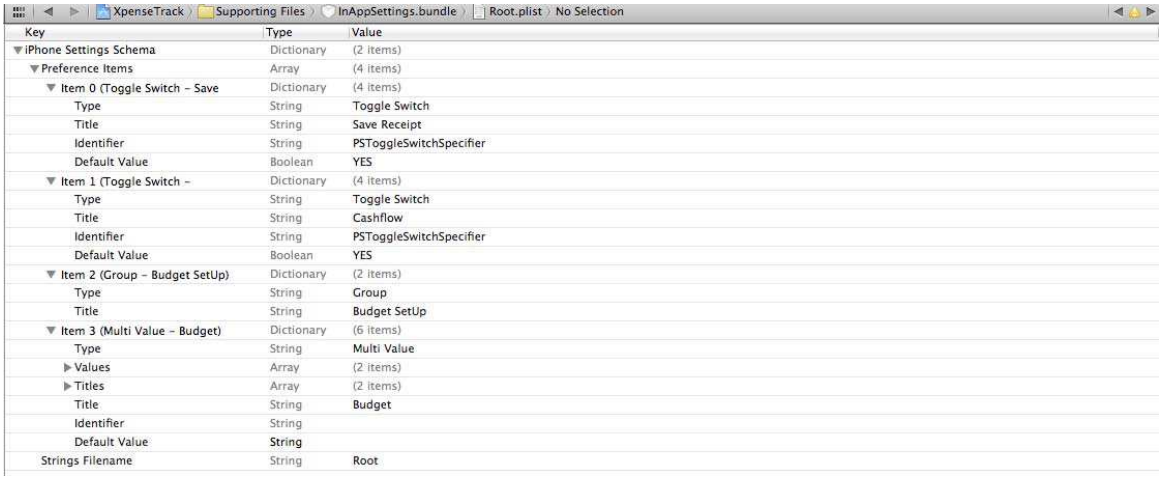

Figure 2.16. Root plist view.

```
        }

UITableViewCell* cell = [tableView dequeueReusableCellWithIdentifier:specifier.type];
        if(nil == cell) {
            cell = [[self newCellForIdentifier:specifier.type] autorelease];
        }

if ([specifier.type isEqualToString:kIASKPSToggleSwitchSpecifier]) {
            cell.textLabel.text = specifier.title;

            id currentValue = [self.settingsStore objectForKey:specifier.key];
            BOOL toggleState;
            if (currentValue) {
                        if ([currentValue isEqual:specifier.trueValue]) {
                                    toggleState = YES;
                        } else if ([currentValue isEqual:specifier.falseValue]) {
                                    toggleState = NO;
                        } else {
                                    toggleState = [currentValue boolValue];
                        }
            } else {
                        toggleState = specifier.defaultBoolValue;
            }
            IASKSwitch *toggle = (IASKSwitch*)cell.accessoryView;
            toggle.on = toggleState;
            toggle.key = specifier.key;
        }
        else if ([specifier.type isEqualToString:kIASKPSMultiValueSpecifier]) {
            cell.textLabel.text = specifier.title;
            cell.detailTextLabel.text = [[specifier titleForCurrentValue:[self.settingsStore
objectForKey:specifier.key] != nil ?
 [self.settingsStore objectForKey:specifier.key] : specifier.defaultValue] description];
            }
```

Figure 2.17. Settings view.

The view consists of two sections. The save Receipt toggle switch allows the user to save the photo of the receipt to the photo gallery or not. Toggling cashflow switch to ON state will enable the user to see the cashflow information on the saved expense/income view otherwise it won't show up the cashflow summary. By default, these switches are turned ON.

The Budget Set up is a multivalue specifier which enables the user to enter the budget information. It is a custom budget whose cycle can be set to every day, every week, every

two weeks, every 4 weeks, every month, every three months, every year. By default, the budget set up is set to OFF and the user can edit it to custom budget and enter the required budget based on the income or the user's needs and requirements. The budget set up in this application is very efficient in order to see how much is the user spending and if it would meet the budget set up or go a level down.

CHAPTER 3

RESULTS

3.1 EXPENSE/INCOME DEMO

When the user feeds the inputs for an expense/income made, a table view is displayed showing the list of expenses/incomes for the particular day and date. Figure 3.1. shows the results for the Expense/Income table view where the red color row clearly displays an expense and a green row specifies an income. The topmost view shows the number of expenses made, the total expense amount and the daily average spending for that particular month. Figure 3.2 shows the income amount, expense amount and the cash flow along with the list of the expenses/income made.

These visual displays are important as the potential user can see what he is spending on, how much he is spending, what he is earning and a cash flow to see the overall progress. All the mathematic calculations are done by the application itself and the user doesn't have to worry about anything.

3.2 DASHBOARD DEMO

Sometimes, the user wants to know how much is he spending weekly, monthly or yearly and the dashboard provides the user with this flexible feature where he can see his expenses and expense amount based on Category, Vendor and payment source. With this feature, the user can definitely look over and cut down on the expenses based on some specific category or vendor. Figure 3.3 and 3.4 shows the dashboard results based on week, month or year along with categories, vendor or payment source. Again, the red color section headers shows expenses and green color section headers represent income made during that time.

Figure 3.3 show the list of expenses, the expense amount and the total expense amount for the month and year as selected by the user based on the vendor and category respectively.

Figure 3.1. Expense/income demo with expense summary.

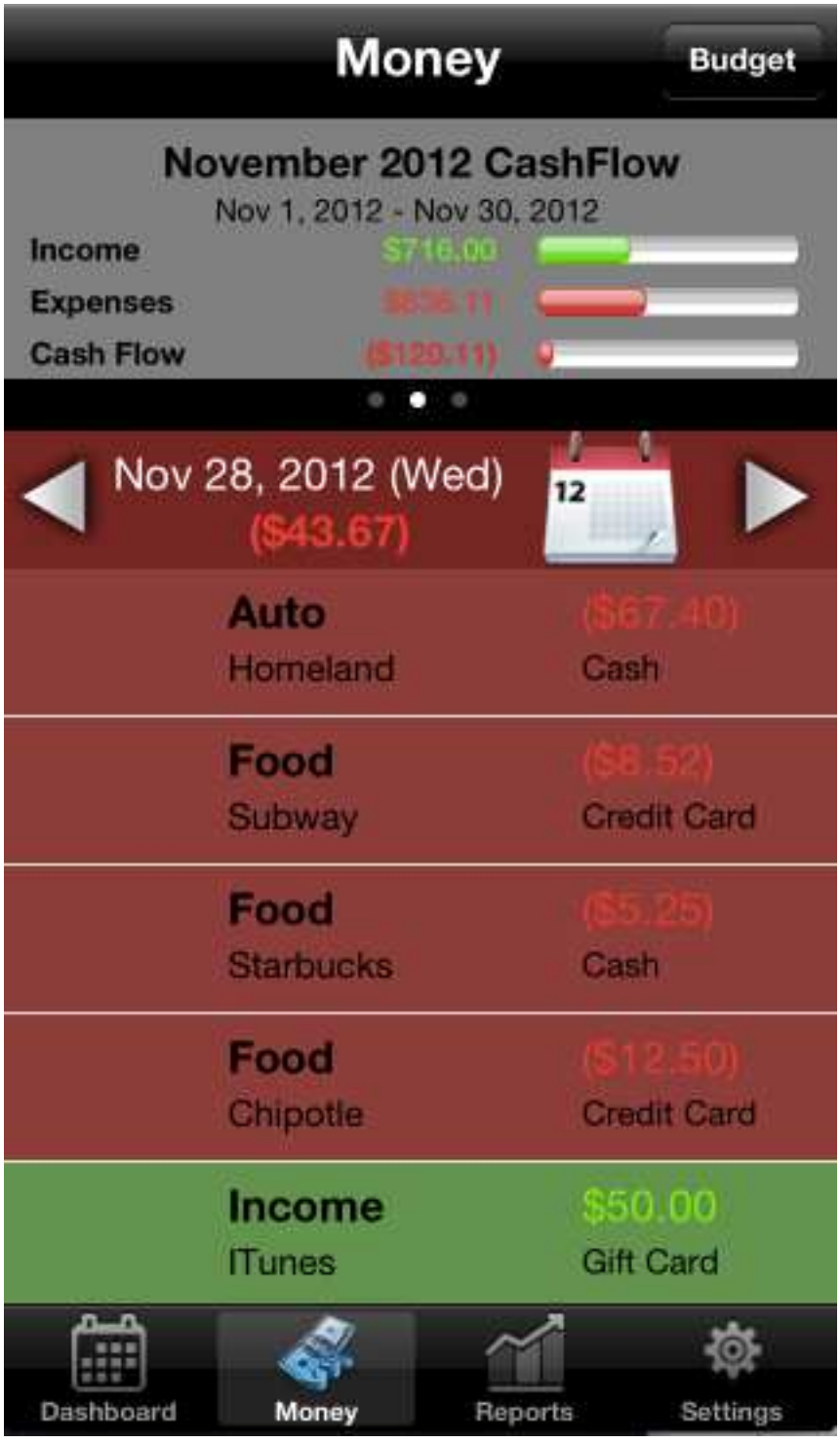

Figure 3.2. Expense/income demo with cashflow.

Figure 3.3. Dashboard demo 1.

Figure 3.4. Dashboard demo 2.

Figure 3.4 show the list of expenses and incomes, the expense/income amount and the total expense/income amount for the week and month as selected by the user based on the category and payment source respectively. The advantage of such approach is the user can keep the expense track activity by cash, credit card or various other payment source methods.

3.3 Reports Demo

The reports section displays the results from the user feeds in three forms- Summary, Bar Graphs and Pie charts. The reports summarizes the total expenses/incomes made based on the user selection from a specific period of time to another period of time.

3.3.1 Summary

The summary, Figure 3.5 shows the total cash flow progress based on the user selected dates and display choice (payment source and vendors).

Figure 3.5. Summary demo 1.

Figure 3.6 shows the total cash flow progress based on the user selected dates and display choice (subcategories and categories).

Figure 3.6. Summary demo 2.

3.3.2 Bar Graphs

The total expense/income amount and it's summary is seen better if shown graphically. A visual representation is beneficial for the user to get the clear estimate where he needs to cut down his expenses. The user selects a time period and display choice in order to see the total expense amount graphically. When the user selects the bar graph button, a pop up gets displayed allowing the user to choose "Expenses" or "Income" in order to see the required results.

Figure 3.7 shows the bar graphs for expenses made during a certain period of time selected by the user based on categories and payment source.

Figure 3.7. Bar graph expense demo.

Each item is represented by it's own individual bar and with it's own specific amount.

Figure 3.8 shows the bar graphs for income made during a certain period of time selected by the user based on subcategories and payment source.

3.3.3 Pie-Charts

Another way of viewing data graphically is by the use of Pie charts that is a very powerful and efficient way to see the data in terms of percentage. Figure 3.9 shows Pie charts for expenses based on payment source and vendor.

For the first part of Figure 3.9, the total expense amount is shown in red color and the specific percentage values represent how much is being spend by cash, check or credit card.

For the second part of Figure 3.9, the percentage values represent how much is spent on which vendor during that particular period of time.

Figure 3.8. Bar graph income demo.

Figure 3.10 shows Pie charts for incomes based on payment source and vendor.

For the first part of Figure 3.10, the total income amount is shown in green color and the specific percentage values represent how much is credited by the vendors such as iTunes etc.

For the second part of Figure 3.10, the percentage values represent the payment source by which the amount is credited during that particular period of time.

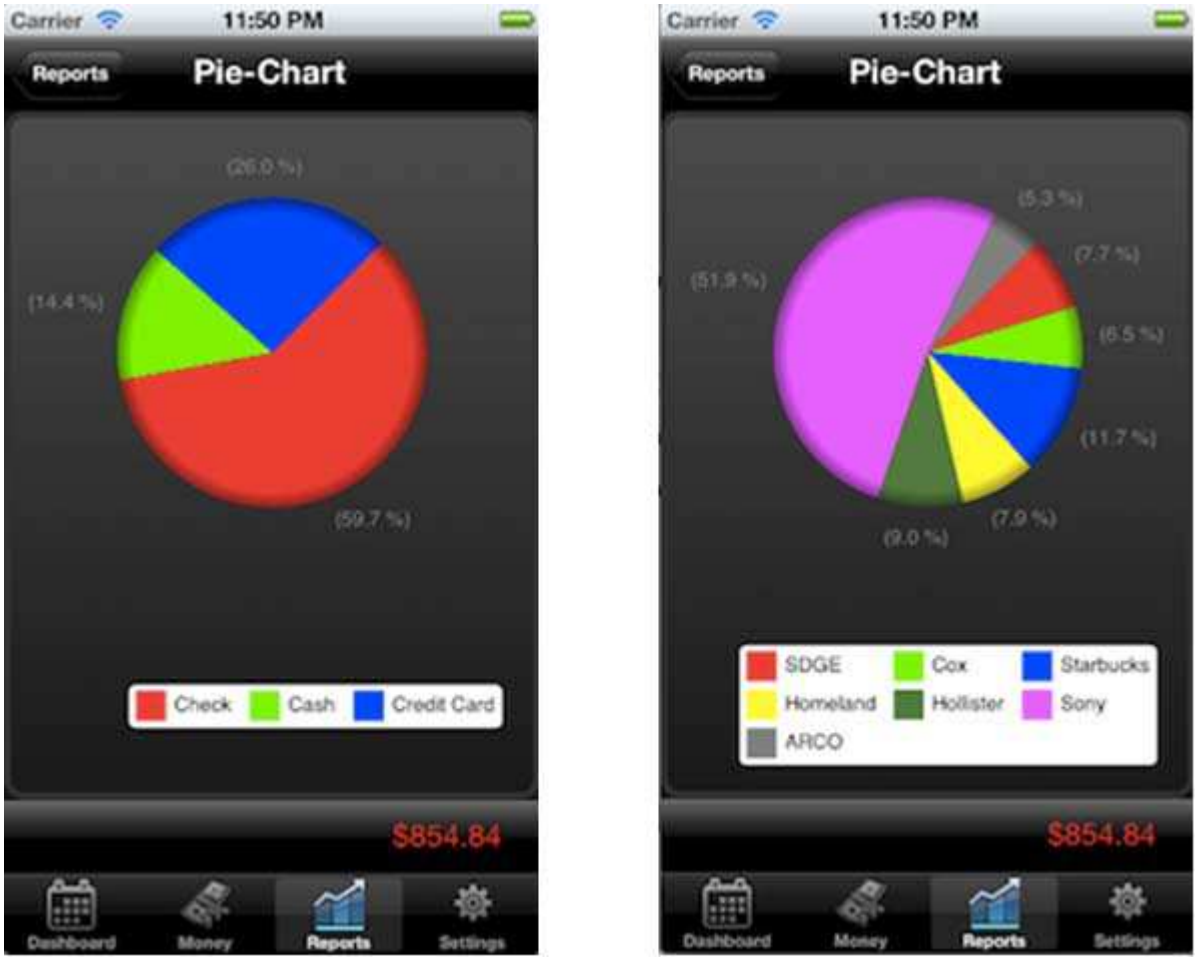

Figure 3.9. Pie chart expense demo.

Figure 3.10. Pie chart income demo.

CHAPTER 4

COMPARISON WITH OTHER MOBILE PLATFORMS

The most common mobile operating systems currently used are iOS, Android, Blackberry OS, Symbian OS and Windows Mobile. Each operating system has it's own unique features and performance areas [18, 19] as discussed in this section.

In terms of User Interface Design, Android's UI is not as smooth or fluid as iOS or WP7 because it runs on bytecode whereas iOS runs on native code. All the UI rendering on the iOS is done on the UI Thread in real time priority whereas android follows PC rendering on the main thread with normal priority. iOS interface design is very intuitive and has got great shiny look with bright colored icons and menus. Apple really focuses on the fine user experience while using the iPhone device. Android has a pull down menu with color schemes whereas blackberry has a swipe up application menu mechanism on the bottom of the screen.

In terms of development environment, every operating system has an IDE, a simulator and some debugging tools. Blackberry and Android requires the use of Java, android developed on Eclipse IDE whereas iOS is done on a Mac with Objective C as the programming language and Xcode as the IDE. WP7 is done using C# language. Objective C++ can be used as well in Xcode in order to develop the applications. Applications that are submitted to Apple takes almost two weeks to be approved and then they become available on the store whereas on Google play, it's just a matter of couple of hours.

In terms of development ease, many devices such as nexus, Samsung, Motorola work on Android OS which have different screen sizes, so the developer has to tweak around and find ways at the run time of the app so that the application user interface looks good on all the different devices that are different in sizes whereas with iPhone having one fixed size, it makes easy for the developers to adjust the frames and sizes. But with the launch of iPhone5, which is taller than the other iPhone devices, developers have to work little more on it so that the graphics elements fix properly.

In terms of performance, Android supports NVIDIA Tegra 2 dual core processor, whereas Blackberry, Symbian and Apple supports dual core processor. However, the new iPhone5 supports Apple A6 processor which is basically a 1.3 Ghz custom apple designed ARMv7 based dual core CPU whereas the previous devices were using Apple A5 processor.

In terms of memory management, iOS 5 provides ARC which means that Objective C takes care of the memory management itself and freeds the developer from the pain of manual memory management by retaining and releasing objects. Android and Blackberry use garbage collection to avoid the memory leaks. The memory leaks in iOS can be detected by various tools such as NSZombies and Instruments provided with the Xcode IDE. The user cannot expand the memory of an iPhone device whereas Android OS can have external memory cards.

In terms of stability, iOS is more stable and less error prone whereas Android is unstable and because of the multitasking nature, the processes can conflict with each other causing crashes.

Every operating system has it's own advantages and disadvantages over the other varying with their hardware, software features, user interface designs, performance issues, memory issues etc.

CHAPTER 5

CONCLUSION

We now present and discuss the limitations of the new product, issues faced, and the remedies to those limitations. While building this expense tracking software tool, the major focus was to make this tool less user intensive and more user productive. The first version of this application is only applicable to the USA. It could have been used in other countries if I would have used currency converters in the application, which I will improvise in the later version. Certain issues were faced while implementing this tool and various important things were kept in mind. For example, the user interface is designed simple yet creative so that the user doesn't face any difficulty in using the software tool and the expense data is persisted on the device even if the user deletes the application from the memory background. Core Data was chosen over SQLite to persist the data which is very beneficial even though the data would reside on the device locally.

iCloud functionality is missing. This would allow data to be saved to a Cloud and shared between various devices. The application lacks the feature of PDF by which the list of expenses and amount could have been displayed in one file in PDF format making it easily readable for the users.

Despite of the mentioned flaws, the application would work as a charm for tracking expenses and provides the user, the flexibility to see the reports in any format (PDF, Bar Graphs, Pie-charts, summary etc.).

Section 5.1 discusses the various limitations in the application and Section 5.2 discusses the future work in order to fix the current problems.

5.1 CURRENT LIMITATIONS

The application is designed in such a simple and straight forward manner that the user faces no problems or difficulties in using this software tool to track the expenses. Still the application suffers from various limitations which doesn't create any obstacle in the current

version as such but these limitations need to be addressed in the next version of the application.

The user can only enter the expense/income amount in USD Currency. So, this application would primarily be used in USA only and cannot be used all over the world because other currencies are not being addressed in this version. The expenses/income can only be tracked in US currency thereby.

The application could have been more user friendly. For example – if the user keeps track of the daily expenses and spends money at Starbucks everyday, he has to enter all the information (merchant, date, amount etc.) himself all over again.

Searching functionality is missing in the current version of the application. Suppose the user wants to search and see the expenses made for a particular category say "Food - Starbucks" for the past 3 months, the user has to scroll through the calendar provided and see it.

The reports in terms of graphs show a comparison of the expenses/incomes separately made on different categories/subcategories. But the graph report doesn't show or evaluate income made v/s expense.

5.2 FUTURE WORK

All the limitations discussed in Chapter 5 should be addressed in the next version of the application so that it can be more enhanced and user friendly.

Provision to add different currencies will be added so that this application is not just limited to USA but also can be used worldwide and the currency converters will be designed and added in order to convert the different currency rates.

In order to make it more user friendly and less user intensive, when the user tries to add the same category or vendor to an expense/income record, a duplicate alert will be presented showing the same category/vendor which the user entered previously for some expense/income and then he can tap on it and the entries will be automatically filled for the current record. For example: the user spends US$ 10 at Starbucks (vendor) on drinks (category) on a particular date and the next day he spends some money at the same place on the same category, then when he tries to write that on the expense details view, a duplicate pop up will be presented showing Starbucks as the vendor and drinks as the category. The

user taps it and it automatically fills up the detail view making the application less user intensive.

A new tab named “Search” will be implemented so that if the user searches for any vendor, category or subcategory by name, he can see the expenses made on that particular search in a table view list with the total number of transactions made and the total expense amount for that search. This would provide a lot more flexibility for the users to track the particular expenses on particular items.

Also, the graph reports show the expenses and income graphs separately in the current version. In the future, a comparison between the income made and expense will be shown graphically providing the user more options to see what they are making and what they are spending accordingly. A PDF feature would be implemented so that the user can see the total expenses/incomes in a much simpler PDF format in one file.

REFERENCES

[1] D. GRAZIANO, *Gartner: Apple leads smartphone sales to new heights*. BGR Media, http://bgr.com/2012/02/15/gartner-apple-leads-smartphone-sales-to-new-heights/, accessed October 2012, February 2012.

[2] M. BROWNLOW, *Smartphone statistics and market share*. Email-Marketing-Reports, http://www.email-marketing-reports.com/wireless-mobile/smartphone-statistics.htm, accessed October 2012, October 2012.

[3] THE WASHINGTON POST, *iPhone 5 sales above 5 million, Apple says, surpassing iPhone 4S*. The Washington Post, http://www.washingtonpost.com/business/technology/iphone-5-sales-above-5-million-apple-says-surpassing-iphone-4s/2012/09/24/8b23322e-0648-11e2-afff-d6c7f20a83bf_story.html, accessed September 2012, September 2012.

[4] C. HEATH AND J. B. SOLL, *Mental budgeting and consumer decisions*, J. Consumer Res., 23 (1996), pp. 40-52.

[5] J. HASTINGS AND J. M. SHAPIRO, *Mental accounting and consumer choice: Evidence from commodity price shocks*. Unpublished report, 2012.

[6] MINT, *Homepage*. Mint, https://www.mint.com, accessed October 2012, n.d.

[7] APPLE, *iOS technology overview*. Apple, http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html, accessed October 2012, n.d.

[8] APPLE, *Key objects of an iOS app*. Apple, http://developer.apple.com/library/IOs/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/AppArchitecture/AppArchitecture.html#//apple_ref/doc/uid/TP40007072-CH3-SW2, accessed October 2012, n.d.

[9] APPLE, *App states and multitasking*. Apple, http://developer.apple.com/library/IOs/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html#//apple_ref/doc/uid/TP40007072-CH4-SW1, accessed October 2012, n.d.

[10] THE OMNI GROUP, *OmniGraffle for Mac*. The Omni Group, http://www.omnigroup.com/products/omnigraffle/, accessed November 2012, n.d.

[11] D. MARK, J. NUTTING AND J. LAMARCHE, *Beginning iPhone 4 Development: Exploring the iOS SDK*, Apress, New York, New York, 2011.

[12] APPLE, *Tab bar controllers*. Apple, http://developer.apple.com/library/ios/#documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/TabBarControllers.html, accessed October 2012, n.d.

[13] APPLE, Introduction to core data programming guide. Apple, http://developer.apple.com/library/mac/#documentation/cocoa/conceptual/coredata/cd

ProgrammingGuide.html#//apple_ref/doc/uid/TP30001200-SW1, accessed October 2012, n.d.

[14] P. ALESSI, *Professional iPhone and iPad Database Application Programming*, Wiley Publishing, Indianapolis, Indiana, 2011.

[15] G. AMBROZIO, *Block-based action sheet*. Cocoanetics, http://www.cocoanetics.com/2012/06/block-based-action-sheet/, accessed October 2012, n.d.

[16] PAVEL, *PMCalendar for iOS*. Cocoa Controls, http://www.cocoacontrols.com/platforms/ios/controls/pmcalendar, accessed October 2012, n.d.

[17] L. VANDAL AND O. GENTZ, *Homepage*. InAppSettingsKit, http://www.inappsettingskit.com/home, accessed October 2012, n.d.

[18] MOBILE INNOVATION, *Smartphones operating systems war*. The Mobile Innovation, http://www.themobileinnovation.net/smartphones-operating-systems-war-android-vs-blackberry-vs-ios-vs-symbian, accessed October 2012, n.d.

[19] T. WOOLEY, *A comparative study of the Android and iPhone operating systems*. University of Central Florida, http://www.cs.ucf.edu/~dcm/Teaching/COP5611Spring2010/Project/TravisWooley-Presentation.pdf, accessed October 2012, April 2010.

APPENDIX

COMPARISON OF MINT V/S XPENSTRAK

Mint works by first creating an user account and then entering your bank and credit card information. Mint web application is a complete package that offers graphical representation like graphs and charts to see the expenses but the Mint iPhone app doesn't show any visual representation. There is no way to track cash activity on Mint, although you can just enter any expense if it's payment source is cash. Also, there is no ability to see transactions from your checking account.

Figure A.1 shows the first view when the Mint iPhone app is launched.

Figure A.1. Mint iPhone app launch screen.

XpensTrak provides the users, the flexibility to enter as well as track the cash activity from any certain period of time based on user selection and there is a provision to see the visual representation in terms of Summary, Bar Graphs and Pie charts as seen from Chapter 3. These few things become important for the user to manage and track the expenses carefully and thus maintain a proper lifestyle based on that.

Figure A.2 shows some screenshots from the Mint iPhone application where I created an account and entered one bank information to compare my expense tracking tool with it.

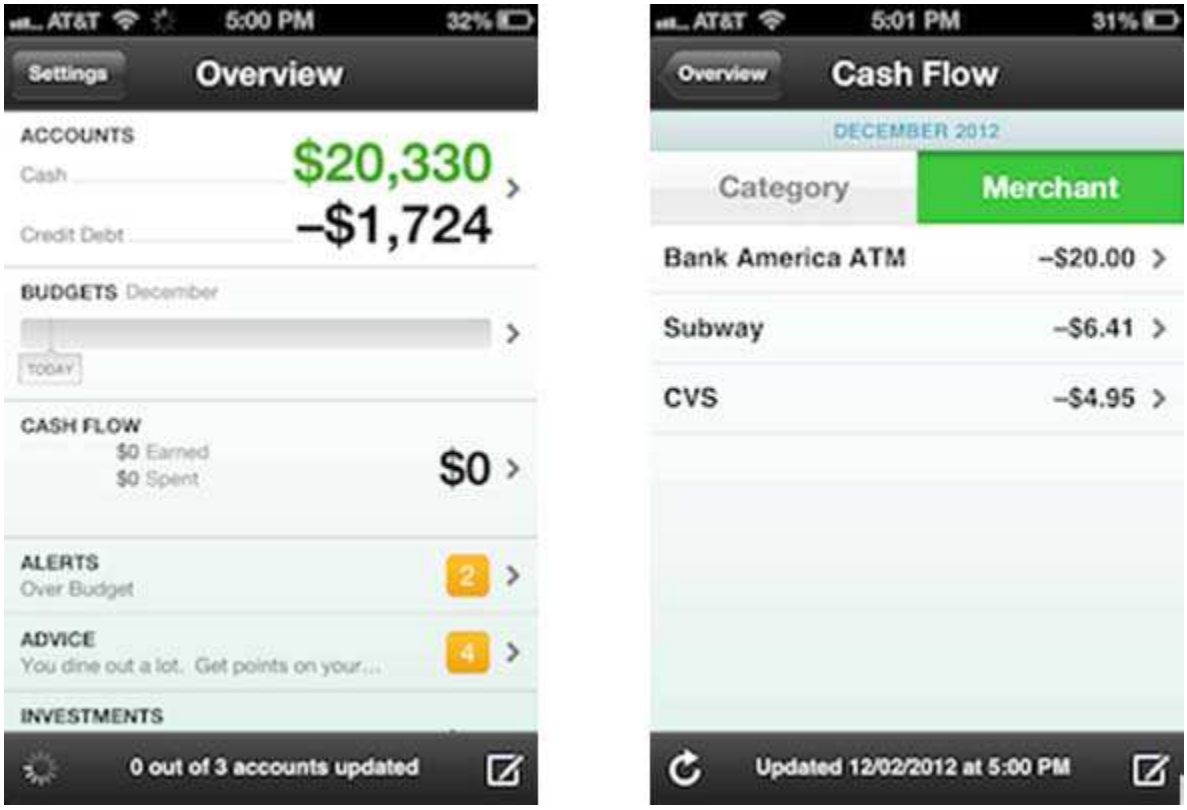

Figure A.2. Mint iPhone app details.