ALHAJJ – HAJJ APP FOR iOS

ADNAN SHAOUT AND SHAHZEB KHAN

The Electrical & Computer Engineering Department,
The University of Michigan – Dearborn,
Dearborn, Michigan 48128, USA.

shaout@umich.edu, and shahzeb@umich.edu

(Received: 05 Nov. 2014; Accepted: 28 Sept. 2015; Published on-line: 30 Apr. 2016)

ABSTRACT: This paper introduces the AlHajj app for iOS which is an interactive guide to Hajj, like an interactive map allowing users to walk through the process of the Hajj to develop a better understanding of the obligations, locations, dates and Hajj activities with the sequence they are performed in. It covers both pre and post Hajj activities. AlHajj has a very simplistic and clean User Interface (UI). The app is also written for maintainability and it is free.

ABSTRAK: Kertas kerja ini memperkenalkan aplikasi AlHajj untuk iOS yang merupakan panduan interaktif bagi ibadah Haji saperti peta interaktif yang menjelaskan pada pengguna mengenai aktiviti-aktiviti dalam mengerjakan Haji dan ia memberi penjelasan yang mendalam mengenai tanggungjawab, lokasi, tarikh dan aktiviti haji mengikut urutan mereka dilakukan . Applikasi ini meliputi aktiviti sebelum dan selepas haji. AlHajj mempunyai antaramuka pengguna yang sangat mudah dan ringkas (UI). Aplikasi ini mudah dislenggarakan dan ia adalah percuma.

KEYWORDS: *Hajj; iOS; Software Tools; Interactive Map; Mobile App; Islam*

1. INTRODUCTION

About three to four million Muslims travel to Mecca every year to perform one of the five pillars of Islam: Hajj (pilgrimage). Hajj involves a series of obligations/rituals that have to be fulfilled/performed over the course of five to six days. It also involves traveling between various sacred locations [1]. Most pilgrims attend training sessions, or go over other material to prepare them for Hajj.

There exist many works done on helping pilgrims during their holy trip to Mecca [2-16]. In today's mobile world, more people have started to use smart-phones to consume information. Therefore usage of mobile apps and mobile websites for information consumption has increased. Some mobile apps pertaining to the obligations of Hajj already exist on the App Store, but we did not find them as interactive, informative and as organized as we would have liked. Some of the apps that were tried were too dull, while some were purely based on linear scrolling through all the steps of Hajj.

The idea of AlHajj is to have a more interactive guide to Hajj, allowing users to walk through the process of the Hajj to develop a better understanding of the obligations, locations, dates and the order in which these rituals are performed. Another feature, such as pre-Hajj customizable checklist can also be beneficial to pilgrims, which is not found in most of the existing Hajj apps.

This paper is organized such that it begins with discussing the state of the art and differentiators for AlHajj in section 2. The feature list of AlHajj is covered in section 3.

Then it discusses the technologies used in the application in section 4. Section 5 sheds some light on the software tools used and implementation details as well as an appendix on how to recreate the AlHajj App for iOS. Also appendix B has the implementation Code for the AlHajj App.The possible future enhancements are discussed in section 6. The paper ends with a conclusion about AlHajj in section 7.

2. STATE OF THE ART

Currently a few applications exist in the App Store that aim at helping people with their Hajj journey.

One such app is Hajj Guide [17]. This is a very basic application. It is based completely on static text and images. It has hard to read text due to the font size and the amount of static text on the screen. There is no interactivity whatsoever.

Another similar app is Complete Hajj Guide [18]. It too suffers from some of the same problems as the Hajj Guide App. It doesn't have any interactive elements either. Also, this app costs $0.99 as opposed to AlHajj, which is free.

Hajj & Umrah [19] is another app that tries to solve the same problem. However, it has the same problems as already discussed for the application above. The information is not very well organized either. There are too many menus to dig through, which makes a user feel that the information could perhaps be more streamlined. On top of all that, this app costs $1.99 as well.

Hajj & Umrah Easy Steps [20] too is a paid app with all static content, currently priced at $0.99. This app has a poor user interface with color combinations so bad that it will take some effort to read information being presented to the user. Also, this app, like the afore-mentioned apps, does not have any interactive elements.

2.1 AlHajj Differentiators (Unique Features)

2.1.1 Interactivity

AlHajj is all about interactivity. Interactivity starts from the pre-hajj preparation checklist to the rituals that are required for Hajj. AlHajj makes users take actions and responds accordingly. This keeps the users in the mode of continues learning and excited. None of the previously mentioned apps exhibit this level of interactivity.

2.1.2 Pre and Post Hajj Coverage

AlHajj is a complete package. It starts with the material required to ensure that a person is absolutely ready to depart for the journey of Hajj, and it also includes motivational messages from the Quran and Hadeeth to ensure that pilgrims stay motivated and on the right path even after they are done with Hajj.

2.1.3 Clean UI

AlHajj has a very simplistic and clean User Interface (UI). The font sizes, arrangement and organization of the text, makes it easily readable. Also, only two colors have been used throughout the app: black and white. This helps ensure that users are focused on the actual material and are not distracted by anything flashy.

2.1.4 Free

- AlHajj costs nothing as compared to the other related apps in the market. Despite the fact that AlHajj has more to offer, it is free.

2.1.5 Maintainability

The ability to easily edit the content in the AlHajj app is a huge plus. All the static content of the app is organized in separate plist and text files, which makes it easy to update any piece of static content. This too gives AlHajj an edge over other competitors.

3. ALHAJJ FEATURE LIST

The major features of AlHajj are:

a) Interactive Checklist
b) Interactive Hajj Map
c) Interactive Contact List
d) Motivational Messages (Chirps)

A brief description of each of these features is as follows:

3.1 Interactive Checklist

This is a checklist of items that need to be accomplished before departing on the journey for Hajj. These items are mostly question like “Have you paid all your debts?” or “Have you written your will?” The users are able to utilize a set of default checklist items. The user can also add more items to the list to suit their needs or to fulfill the requirements specific by their country of origin. They can mark items as done and delete the ones that do not apply to them. Figure 1 shows screenshots of the interactive checklist.

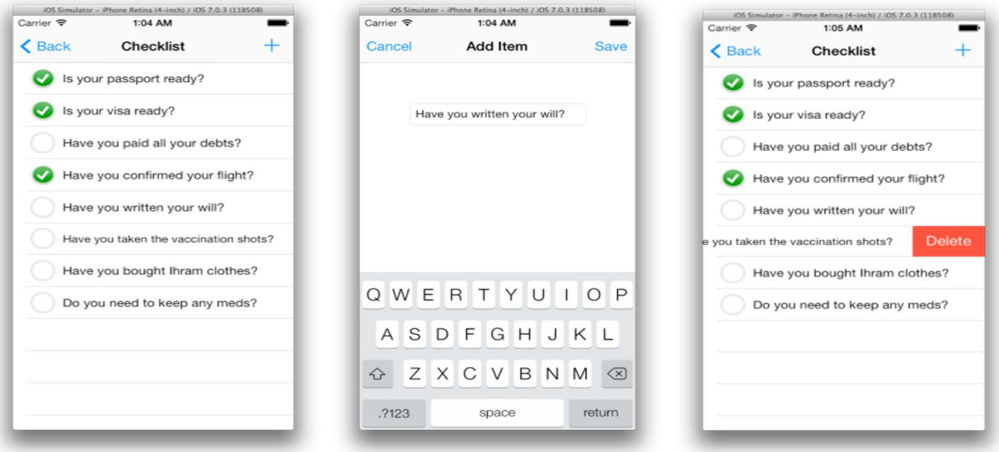

Fig. 1: Screenshots of the interactive checklist.

3.2 Interactive Hajj Map

AlHajj lets users learn about the procedures of Hajj through an interactive map that they can walkthrough. The map lets users see what needs to be done, at what date and at what location. It allows users to go through each location that needs to be visited in the correct order. It also allows the user togo back to previous locations to see what they have already done.

While at a location they can tap the location, or the “See Details” button to view all the details of procedures that need to be performed at that location. Once users have finished walking through the entire process, the map asks them if they want to start over. Users can

always reset the map by clicking the "Reset Map" button, even when in the middle of the walkthrough and start over. Figure 2 shows the interactive map screenshots.

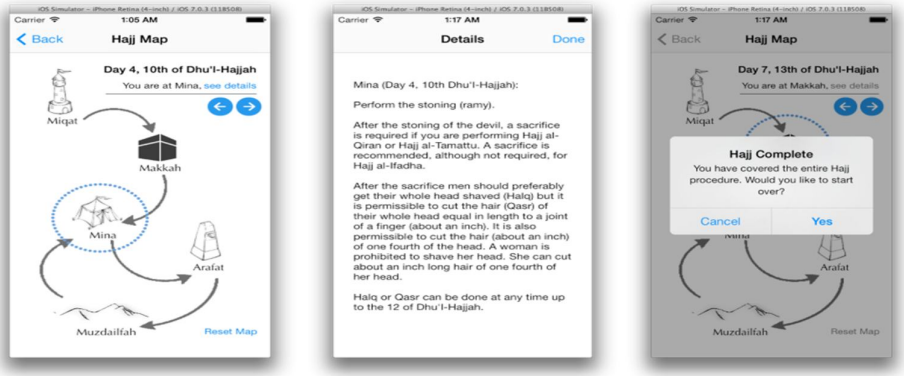

Fig. 2: Interactive Map Screenshots.

3.3 Interactive Contact List

This is a list of all of the important contacts that the users may need during their stay in Saudi Arabia for Hajj. The set of default contacts include Saudi Police Department, Fire Department, Highway Patrol, Ambulance, etc. Users can add more contacts to this list as well to suit their personal needs and requirements. However, users are only allowed to delete contacts that they have themselves added, and not the default ones.

Another great ability related to the interactive contacts list, is the ability to dial a phone number from within AlHajj. The user doesn't need to exit the app to dial a number. All the user has to do is to tap the number that he/she wants to dial in the AlHajj contact list. The app will then ask for confirmation, and the number will be dialed. This is a great convenience, and will definitely come in handy if an emergency situation is encountered. Figure 3 shows the Interactive Contact List Screenshots.

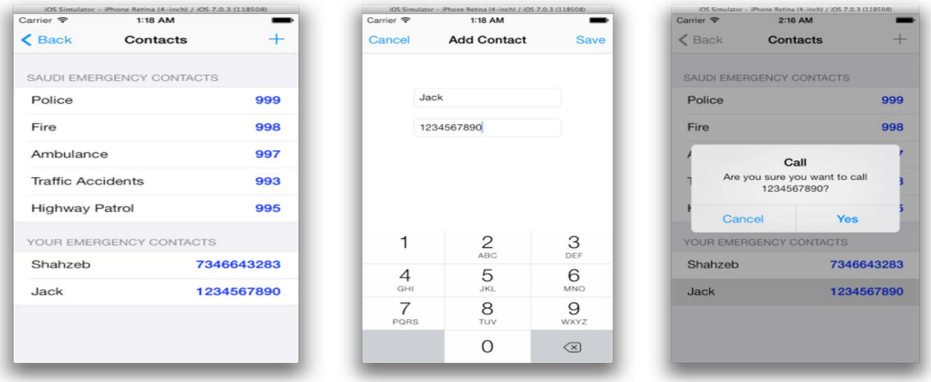

Fig. 3: Interactive Contact List Screenshots.

3.4 Motivational Messages (Chirp)

AlHajj allows users to setup a time to receive motivational messages on their phone. These messages can be Quranic quotations, a Hadeeth or anything else, aimed at boosting the religious spirit of the users. There are three hundred and sixty six slots for these

messages. Each slot corresponds to a day of the year. Hence, users will receive a different motivational message each day. Users can always go back and change the time they want to receive these messages. They can also completely turn off these messages if they want. Figure 4 shows motivational messages screenshots.

Fig. 4: Motivational messages screenshots.

4. SOFTWARE TOOLS USED

4.1 iOS SDK, Objective C, Xcode

This is the most important and core package of tools that was needed for this app. Applications for iOS cannot be developed without using the iOS SDK [21], which is based on Objective-C [22]. Xcode [23] is Apple's integrated development environment for developing apps for mobile (iOS) and desktop (OS X) devices.

4.2 Bitbucket / GitX

As the codebase for this paper started to grow, it was impossible to keep track of the changes manually. Therefore we hosted the code in a repository on Bitbucket [24]. It is free, which allows version control using mercurial and git.

GitX [25] was also used locally. It is a client for Mac, which helps in keeping track of all the changes on the local and remote branches of code.

5. IMPLEMENTATION DETAILS

5.1 Interactive Checklist

To implement the checklist, the UI for the checklist view is implemented using the storyboard. The view is sub-classed from the UITableView class to get all the necessary attributes of the native iOS table view. Each row of the table is an item of the checklist.Prototype cells have been used to customize each row of the cell. Each prototype cell has an image view embedded inside it towards the leftand a text view adjacent to it. The image view is used to show and hide the green checkmark, which signifies the completion of a task. The text view is used to display the actual text for the checklist item.

Table view delegate methods have been implemented to capture the behavior of tapping a table row, which turns the checkmark on or off.Also, the swiping left on a table row will bring up an option to delete it. Corresponding delegate methods have been implemented to handle deletion.

All the items present in the list by default are populated by using a plist file present in the code base. Items can be added or removed from it to alter the default list of items.

In order to add new items to this list, a separate add view is implemented. It is sub-classed from UIView, instead of UITableView, since this is a simple view and not a table view. The UI is implemented using the storyboard. Anything saved here is added to the original list.

5.2 Interactive Hajj Map

To implement the interactive Hajj map, the UI for the map view has been laid out by using the storyboard. The view is sub-classed off of a UIView. All the image views are positioned in the storyboard. These image viewshave been used for all locations and for circles around them, like Mecca, Mina, etc. Images have also been used for all arrows, representing traveling between the various locations. Text views are used to represent all the text on the map view. The next/previous, see details and reset map buttons are implemented using iOS native UIButton.

When the view is initialized all arrows are kept hidden, all locations are visible and only onecircle is displayed, which is on the initial location (Miqat).The entire sequence of traveling is loaded into arrays. For example, all arrows are ordered into an array, and all labels for each location are ordered into an array as well. When the next button is tapped, logic is triggered to move to the next element in the various arrays. Similarly, tapping previous button triggers logic to go to the previous element in the various arrays. This affects what is shown and what is hidden, and thus forms the interactive map.

In order to view procedure details for a particular location, a separate detail view has been implemented. It is sub-classed from UIView. There are about nine location points that require reading details. However, implementing nine separate views to handle this would have been pretty tough to maintain. Therefore, only one detail view has been implemented with nine text files containing the details. The detail view dynamically loads one of these nine files, based on the current location of the user.

5.3 Interactive Contact List

To implement the contact list, the UI for the contact list view is implemented using the storyboard. The view is sub-classed from the UITableView class to get all the necessary attributes of the native iOS table view. Each row of the table is an item of the contact list. Prototype cells have been used to customize each row of the cell. Each prototype cell has two text views. The text view on the left is used for the name of the contact, and text view on the right is used to display the phone number for the corresponding contact.

The table view is divided into two sections to accommodate the two kinds of contacts: default contacts and user contacts. The default contacts are stored in a static plist file that is part of the code base. This file can be edited to alter the default contact list. Users are also able to add their own contacts to this list. These contacts appear in a separate section on the table view and can be deleted.

In order to add new items to this list, a separate add view is implemented. It is sub-classed from UIView, instead of UITableView, since this is a simple view and not a table view. The UI is implemented using the storyboard. Anything saved here is added to the original list.

Table view delegate methods have been implemented to capture the behavior of tapping a table row, which triggers a UIAlertView asking if the user wants to make a phone call to the tapped contact. If the user taps “Yes”, a phone call is made from with the AlHajj app.

5.4 Motivational Messages

To implement the motivational messages, the UI for the chirps view has been laid out by using the storyboard. The view is sub-classed off of a UIView. UIDatePicker has been used for date selection, and a UISwitch has been used to turn these motivational messages on or off. All of these motivational messages are stored in a separate plist file that is part of the code base.

UILocalNotifications have been used for scheduling and triggering these messages. Sinceeach day is supposed to have its unique motivational message, it requires that three hundred and sixty six notifications be scheduled to trigger the messages every day of the year.

Apple does not allow more than 64 local notifications to be scheduled at a time by an application [26], which posed a serious problem. The solution around this was to schedule sixty notifications each time the app is launched. This schedules enough messages for the nextcouple of months, with a single launch of the app.

Appendix A has the recreation of the AlHajj App for iOS. Appendix B has the implementation Code for the AlHajj App.

6. FUTURE ENHANCEMENTS

6.1 Geo-location

One important feature that can be added to this app is the ability to track the location of fellow group/family members using GPS, since it is easy to get lost in a crowd of four million people.

6.2 Audio for Arabic Du'as

It will be useful to add audio for Arabic Du'as in the app, which will open another avenue of learning for the users.

6.3 More Visual Elements

More visual elements can be added to the app in the future, like simple images or animations, to graphically explain some of the procedures of Hajj.

7. CONCLUSION

AlHajj is an App that has a more interactive guide to Hajj, like an interactive map allowing users to walk through the process of the Hajj to develop a better understanding of the obligations, locations, dates and sequence they are performed in. The App also has a unique feature which is a pre-Hajj customizable checklist. This feature can be beneficial to pilgrims, which is not found in most of the existing Hajj apps. Hence, AlHajj is a clear winner when compared to the other related apps in the market today.

REFERENCES

[1] Hajj General Information [http://www.hajinformation.com/flashdaybyday.htm]

[2] Geabel A, Jastaniah K, Abu Hassan R, Aljehani R, Babadr M, Abulkhair M. (2014) Pilgrim Smart identification using RFID technology (PSI). Marcus A. (Ed.): DUXU 2014, Part III, LNCS 8519, Springer International Publishing, pp. 273–280.

[3] Mohandes M, Haleem MA, Kousa M, Balakrishnan K. (2013) Pilgrim tracking and identification using wireless sensor networks and GPS in a mobile phone. Arab J Sci Eng, 38:2135-2141.
[4] Taileb M, Al-Ghamdi E, Al-Ghanmi N, Al-Mutari A, Al-Jadani K, Al-Ghamdi M, Al-Mutari A. (2014) Manasek AR: A location-based augmented reality application for Hajj and Umrah. Shumaker R and Lackey S. (Eds.): VAMR 2014, Part II, LNCS 8526, Springer International Publishing, pp. 134-143.
[5] Mitchell RO, Rashid H, Dawood F, AlKhalidi A. (2013) Hajj crowd management and navigation system people tracking and location based services via integrated mobile and RFID systems. IEEE conference 978-1-4673-5285-7/13.
[6] Ali MAT, Berri J, Zemerly MJ. (2008) Context aware mobile Muslim companion. CSTST October 27-31, 2008, Cergy-Pontoise, France.
[7] Hamhoum F, Kray C. (2012) Supporting pilgrims in navigating densely crowded religious sites. Pervasive Ubiquity Computers, 16:1013-1023.
[8] Muaremi A, Tr¨oster G, Seiter J, Bexheti A. (2013) Monitor and understand pilgrims: Data collection using smart phones and wearable devices. UbiComp'13, September 8–12, 2013, Zurich, Switzerland.
[9] Zeki AM, Alsafi H, Nassr RM, Mantoro t. (2012) A mobile dictionary for pilgrims. The 2012 International Conference on Information Technology and e-Services.
[10] Mohandes MA. (2012) Near field communication for pilgrim services. The 8th International Conference on Computing Technology and Information Management (ICCM).
[11] Malak Osman and Adnan Shaout (2015), "Overview of Mobile Help for Performing Hajj Rituals", the International Journal of Emerging Technology & Advanced Engineering (ISSN 2250-2459, ISO 9001:2008 Certified Journal), Volume 5, Issue 10, October 2015.
[12] Malak Osman and Adnan Shaout (2014), "Hajj Guide Systems - Past, Present and Future", the International Journal of Emerging Technology & Advanced Engineering (ISSN 2250-2459, ISO 9001:2008 Certified Journal), Volume 4, Issue 8, August 2014.
[13] Ali Alao, Adnan Shaout, Malak Osman (2015), "Tawaf Counting", the proceedings of the 3nd International Conference on Islamic Applications in Computer Science and Technology (IMAN 2015) 1-3 October 2015 Konya, Turkey.
[14] Malak Osman, Adnan Shaout and M. Mohandes (2015), "Easy Hajj Applications Track & Educate Pilgrims", the proceedings of the 3nd International Conference on Islamic Applications in Computer Science and Technology (IMAN 2015) 1-3 October 2015 Konya, Turkey.
[15] Malak Osman and Adnan Shaout (2015), "Pilgrim Communication Using Mobile Phones", the proceedings of the ICCTS 2015, Bandar Seri Begawan, Brunei June 2015.
[16] Malak Osman and Adnan Shaout (2015), "Towards Developing an Intelligent Hajj Guide System – Pilgrim Tracking and Identification Using Mobile Phones", the proceedings of the 7th International Conference on Information Technology (ICIT2015), Amman, Jordan May 2015.
[17] Hajj Guide App by ImranQureshi.com, 2011 [https://itunes.apple.com/us/app/hajj-guide/id473635756]
[18] Complete Hajj Guide App by Cyber Designz, 2012 [https://itunes.apple.com/us/app/complete-hajj-guide/id560955425]
[19] Hajj & Umrah App by AMC Apps, 2013 [https://itunes.apple.com/us/app/hajj-umrah/id471282657?mt=8]
[20] Hajj & Umrah Easy Steps App by Pearls Productions UK, 2011 [https://itunes.apple.com/app/id400656429]
[21] iOS SDK [https://developer.apple.com/technologies/ios/]
[22] Objective C [https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html]
[23] Xcode [https://developer.apple.com/xcode/]
[24] Bitbucket [https://bitbucket.org/]
[25] GitX [http://gitx.frim.nl/]

[26] iOS Local Notifications [https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/WhatAreRemoteNotif.html]

Appendix A: Recreating AlHajj App for iOS

Here are the steps for recreating AlHajj app for iOS.

1. Launch Xcode using Launchpad.

2. Click “Create a new Xcode project”.

3. Select “Single View Application” Template as shown in Fig. a1.

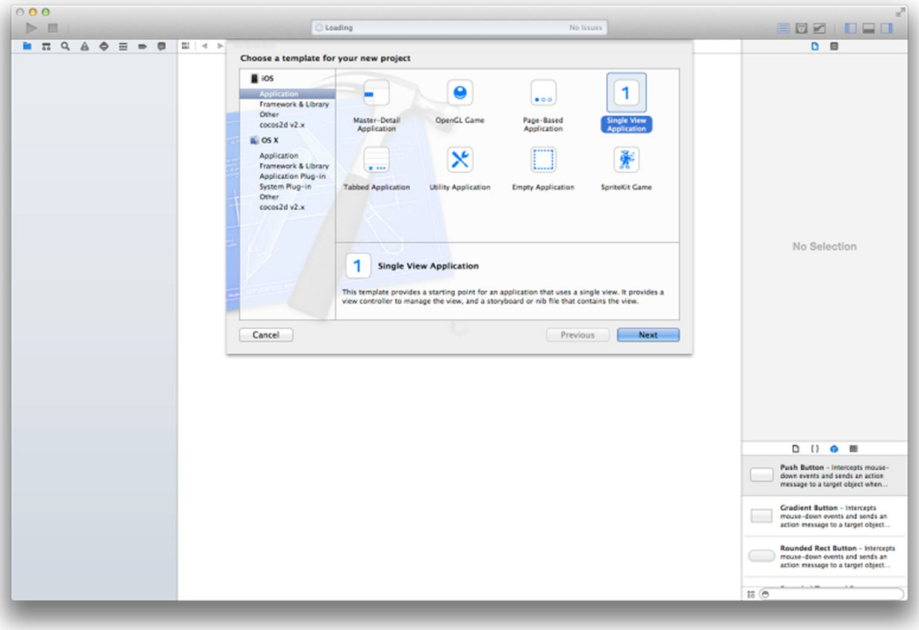

Fig. a1: Design template.

4. Click “Next”.

5. Fill in the name of the app.

6. Select the destination folder for the project.

7. Click “Create” as shown in Fig. a2 and a3.

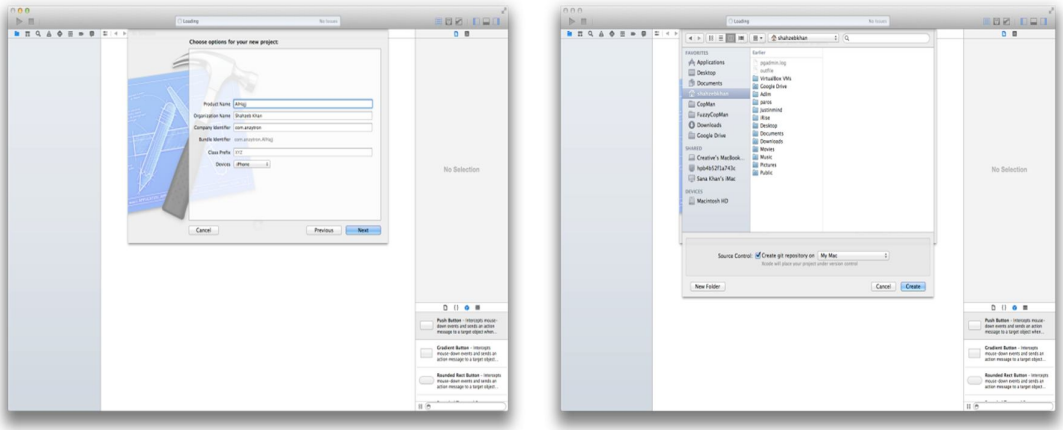

Fig. a2: Creating a project.

Checklist Development:

1. Create a new plist file called Checklist.plist to hold default items, using the file menu shown in Fig. a3.

Key	Type	Value
▼ Root	Array	(8 items)
▼ Item 0	Dictionary	(1 item)
Is your passport ready?	Boolean	NO
▼ Item 1	Dictionary	(1 item)
Is your visa ready?	Boolean	NO
▼ Item 2	Dictionary	(1 item)
Have you paid all your debts?	Boolean	NO
▼ Item 3	Dictionary	(1 item)
Have you confirmed your flight?	Boolean	NO
▼ Item 4	Dictionary	(1 item)
Have you written your will?	Boolean	NO
▼ Item 5	Dictionary	(1 item)
Have you taken the vaccination shots?	Boolean	NO
▼ Item 6	Dictionary	(1 item)
Have you bought Ihram clothes?	Boolean	NO
▼ Item 7	Dictionary	(1 item)
Do you need to keep any meds?	Boolean	NO

Fig. a3: The file menu for the check list.

2. Create a new checklist view as a table view of prototype cells using the storyboard as shown in Fig. a4.

Fig. a4: Prototype cells.

3. Create a new add item view with a UITextfield and toolbar items, using the storyboard as shown in Fig. a5.

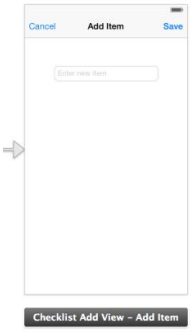

Fig. a5: An add item view.

4. Create a class to contain all code and logic for the checklist view.

5. Create a class to contain all code and logic for the checklist add view.

6. Connect all IBOutlets.

7. Implement view lifecycle methods (viewDidLoad, viewWillAppear, etc).

8. Implement table view delegate methods.

9. Implement methods to read from and write to the plist file.

10. Implement methods to pass data between the two views.

Hajj Map Development:

1. Create text files to store Hajj procedure details, using the file menu.

2. Create a new map view using the storyboard as shown in Fig. a6.

3. Use UIImageViews for all arrows and locations, and UIButtons for circles.

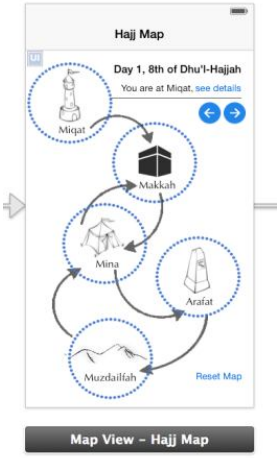

Fig. a6: A new map view using the storyboard.

4. Create a new detail view with a UITextView and toolbar items, using the storyboard as shown in Fig. a7.

Fig. a7:Creating a new detail view.

5. Create a class to contain all code and logic for the map view.

6. Create a class to contain all code and logic for the detail view.

7. Connect all IBOutlets.

8. Implement logic to load all items into arrays.

9. Implement logic to traverse back and forth between items of these arrays.

10. Implement logic to show and hide items based on the selected item in an array.

11. Implement logic to dynamically load procedure details into the detail view.

Contact List Development:

1. Create a plist file called Contacts.plist to store all default contacts, using the file menu as shown in Fig. a8.

Key	Type	Value
▼ Root	Array	(5 items)
▼ Item 0	Dictionary	(1 item)
Police	String	999
▼ Item 1	Dictionary	(1 item)
Fire	String	998
▼ Item 2	Dictionary	(1 item)
Ambulance	String	997
▼ Item 3	Dictionary	(1 item)
Traffic Accidents	String	993
▼ Item 4	Dictionary	(1 item)
Highway Patrol	String	995

Fig. a8: The file menu for the contact list.

2. Create a new contacts view as a table view of prototype cells using the storyboard as shown in Fig. a9.

Fig. a9: Creating a new contacts view as a table view of prototype cells.

3. Create a new add item view with UITextFields and toolbar items, using the storyboard as shown in Fig. a10.

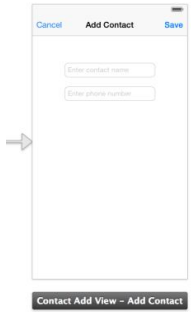

Fig. a10: Creating a new add item view.

4. Create a class to contain all code and logic for the contacts view.

5. Create a class to contain all code and logic for the contact add view.

6. Connect all IBOutlets.

7. Implement view lifecycle methods (viewDidLoad, viewWillAppear, etc).

8. Implement table view delegate methods.

9. Implement methods to read from and write to the plist file.

10. Implement methods to pass data between the two views.

11. Implement method to trigger a phone call when a row is tapped.

Motivational Messages Development:

1. Create a plist file called Chirps.plist to store all default chirps, using the file menu as shown in Fig a11.

Key	Type	Value
▼ Root	Array	(368 items)
Item 0	String	This is chirp number 0
Item 1	String	This is chirp number 1
Item 2	String	This is chirp number 2
Item 3	String	This is chirp number 3
Item 4	String	This is chirp number 4
Item 5	String	This is chirp number 5
Item 6	String	This is chirp number 6
Item 7	String	This is chirp number 7
Item 8	String	This is chirp number 8
Item 9	String	This is chirp number 9
Item 10	String	This is chirp number 10
Item 11	String	This is chirp number 11
Item 12	String	This is chirp number 12
Item 13	String	This is chirp number 13
Item 14	String	This is chirp number 14

Fig. a11: Creating a plist file called Chirps.plist to store all default chirps.

2. Create a new chirp's view with a UIDatePicker and a UISwitch, using the storyboard as shown in Fig. a12.

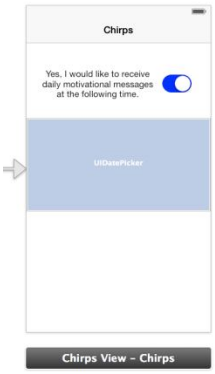

Fig. a12: Creating a new chirp's view with a UIDatePicker and a UISwitch.

3. Create a class to contain all code and logic for the chirps view.

4. Connect all IBOutlets.

5. Implement UIPickerView delegate methods.

6. Implement scheduling of notifications using UILocalNotification.

7. Implement logic to enable/disable notifications based on the UISwitch's state.

Main Menu Development:

1. Create a new main menu view with UIButtons and UILabels, using the storyboard as shown in Fig. a13.

Fig. a13: Creating a new main menu view with UIButtons and UILabels.

2. Use Control + Drag to connect each button to the respective view.

Navigation Development:

1. Control + Drag from the source UI element to destination view element to trigger navigation from a UI element to a view.

2. Once you leave the mouse click on the destination you will presented with various options. Pick "modal" if you are connecting a detail view to a parent view (e.g. Contacts view to contact adds view). Pick "push" for regular navigation between screens.

3. Once you are done setting up all navigation, your storyboard should like Fig. a14.

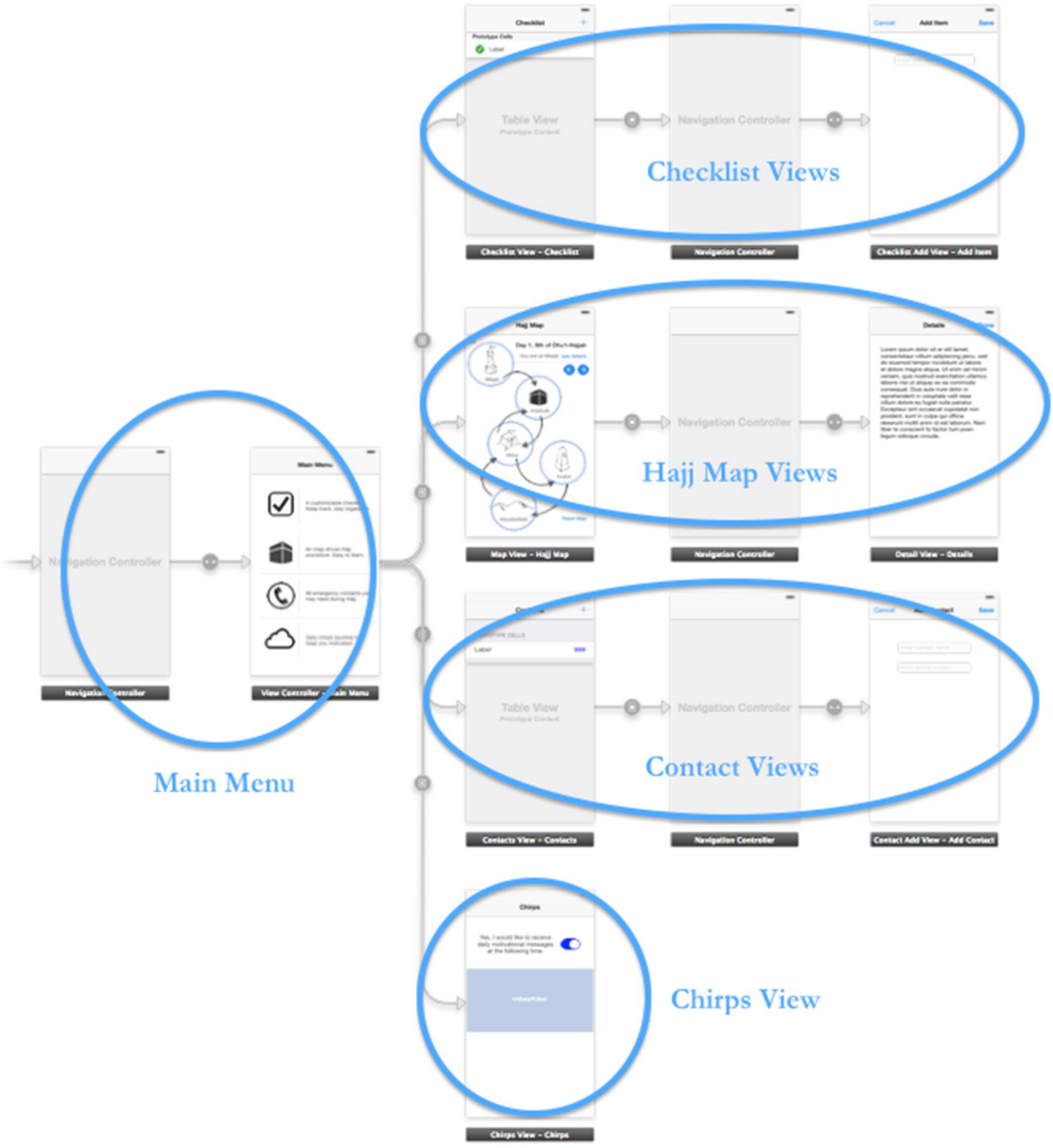

Fig. a14: Storyboard.

Appendix B – Implementation Code for the AlHajj App

```
//
//  ChecklistView.h
//  AlHajj
//
//  Created by Shahzeb Khan on 10/9/13.
//  Copyright (c) 2013 Shahzeb Khan. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "ChecklistAddView.h"
#import "ChecklistAddViewDelegate.h"
#import "CheckListCell.h"
#import "ChecklistItem.h"

@interface ChecklistView : UITableViewController <ChecklistAddViewDelegate> {

    NSMutableArray *checklistItems;
}

@property (nonatomic, retain) NSMutableArray *checklistItems;

@end
```

```
//
//  ChecklistView.m
//  AlHajj
//
//  Created by Shahzeb Khan on 10/9/13.
//  Copyright (c) 2013 Shahzeb Khan. All rights reserved.
//

#import "ChecklistView.h"

@interface ChecklistView ()

@end

@implementation ChecklistView

@synthesize checklistItems;

#pragma mark - View lifecycle

- (id)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self) {
        // Custom initialization
    }
    return self;
}


- (void)viewDidLoad
{
    [super viewDidLoad];

    // Uncomment the following line to preserve selection between presentations.
    // self.clearsSelectionOnViewWillAppear = NO;

    // Uncomment the following line to display an Edit button in the navigation bar for this view controller.
    // self.navigationItem.rightBarButtonItem = self.editButtonItem;

    self.checklistItems = nil;
    self.checklistItems = [[NSMutableArray alloc] init];

    if (![[NSUserDefaults standardUserDefaults] boolForKey:@"loadedDefaultCheckList"]) {
        [self readDefaultChecklist];
        [[NSUserDefaults standardUserDefaults] setBool:YES forKey:@"loadedDefaultCheckList"];
    } else {
        [self readUserChecklist];
    }
}

- (void)viewWillAppear:(BOOL)animated {

    [self.tableView reloadData];
}

- (void)viewWillDisappear:(BOOL)animated {

    [self writeChecklistToPlist];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Return the number of sections.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // Return the number of rows in the section.
    return [self.checklistItems count];
}
```

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"CheckListCell";
    CheckListCell *cell = (CheckListCell *)[tableView dequeueReusableCellWithIdentifier:CellIdentifier];

    if (cell == nil) {
        cell = [[CheckListCell alloc] initWithStyle:UITableViewCellStyleDefault  reuseIdentifier:CellIdentifier];
    }
    // Configure the cell...
    ChecklistItem *item = self.checklistItems[indexPath.row];
    cell.checkListItem.text = item.task;
    if (item.isDone) {
        cell.checkMark.image = [UIImage imageNamed:@"greenCheck.png"];
    } else {
        cell.checkMark.image = [UIImage imageNamed:@"emptyCheck.png"];
    }
    cell.selectionStyle = UITableViewCellSelectionStyleNone;

    return cell;
}

// Override to support editing the table view.
- (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        // Delete the row from the data source
        [self.checklistItems removeObjectAtIndex:indexPath.row];
        [self writeChecklistToPlist];

        [tableView deleteRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationFade];
    }
    else if (editingStyle == UITableViewCellEditingStyleInsert) {
        // Create a new instance of the appropriate class, insert it into the array, and add a new row to the table view
    }
}

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

    CheckListCell *cell = (CheckListCell *)[self.tableView cellForRowAtIndexPath:indexPath];
    [self toggleCheckMarkForCell:cell];
    [self toggleDoneAtIndex:indexPath.row];
}

#pragma mark - Navigation

// In a story board-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
    if ([segue.identifier isEqualToString:@"AddItem"]) {
        UINavigationController *navigationController = segue.destinationViewController;
        ChecklistAddView *checkListAddView = [[navigationController viewControllers] objectAtIndex:0];
        checkListAddView.delegate = self;
    }
}


#pragma mark - Navigation

// In a story board-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
    if ([segue.identifier isEqualToString:@"AddItem"]) {
        UINavigationController *navigationController = segue.destinationViewController;
        ChecklistAddView *checkListAddView = [[navigationController viewControllers] objectAtIndex:0];
        checkListAddView.delegate = self;
    }
}

#pragma mark - Checklist / cell manipulation

- (void)toggleCheckMarkForCell:(CheckListCell *)cell {

    if (cell.checkMark.image == [UIImage imageNamed:@"greenCheck.png"]) {
        cell.checkMark.image = [UIImage imageNamed:@"emptyCheck.png"];
    } else {
        cell.checkMark.image = [UIImage imageNamed:@"greenCheck.png"];
    }
}

- (void)toggleDoneAtIndex:(int)index {

    int count = 0;
    for (ChecklistItem *item in self.checklistItems) {
        if (count == index) {
            item.isDone = !item.isDone;
        }
        count++;
    }
}

- (void)addTaskToChecklistItems:(NSString *)task {

    ChecklistItem *item = [[ChecklistItem alloc] initWithTask:task andDone:NO];
    [self.checklistItems addObject:item];
}

#pragma mark - Checklist add view delegate methods

- (void)checklistAddViewDidCancel {

    [self dismissViewControllerAnimated:YES completion:nil];
}

- (void)checklistAddViewControllerDidSave:(NSString *)task {

    [self addTaskToChecklistItems:task];
    [self writeChecklistToPlist];

    [self dismissViewControllerAnimated:YES completion:nil];
}
```

```
#pragma mark - Read / write checklist

- (void)readUserChecklist {

    [self readChecklistAtPath:[self pathForUserChecklist]];
}

- (void)readDefaultChecklist {

    [self readChecklistAtPath:[self pathForDefaultChecklist]];
}

-(void)readChecklistAtPath:(NSString *)path {

    NSArray *array = [NSArray arrayWithContentsOfFile:path] ;

    for (NSDictionary *dict in array) {
        for (id key in dict) {
            NSString *task = key;
            BOOL isDone = [[dict objectForKey:key] boolValue];
            ChecklistItem *item = [[ChecklistItem alloc] initWithTask:task andDone:isDone];
            [self.checklistItems addObject:item];
        }
    }
}

- (void)writeChecklistToPlist {

    NSMutableArray *array = [[NSMutableArray alloc] init];

    for (ChecklistItem *item in self.checklistItems) {
        NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
        [dict setValue:[NSNumber numberWithBool:item.isDone] forKey:item.task];
        [array addObject:dict];
    }

    [array writeToFile:[self pathForUserChecklist] atomically:YES];
}


- (NSString *)pathForUserChecklist {

    NSString *documentsDirectory = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) firstObject];
    return [NSString stringWithFormat:@"%@/Checklist.plist", documentsDirectory];
}

- (NSString *)pathForDefaultChecklist {

    return [[NSBundle mainBundle] pathForResource:@"Checklist" ofType:@"plist"];
}

@end
```

Fig. b1: Interactive Checklist Code.

```
//
//  MapView.h
//  AlHajj
//
//  Created by Shahzeb Khan on 12/7/13.
//  Copyright (c) 2013 Shahzeb Khan. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "DetailView.h"

@interface MapView : UIViewController <UIAlertViewDelegate> {

    NSArray *allArrows;
    NSArray *allCircles;
    NSArray *allDayLabels;
    NSArray *allPositionLabels;

    IBOutlet UILabel *dayLabel;
    IBOutlet UILabel *positionLabel;

    IBOutlet UIButton *miqatCircle;
    IBOutlet UIButton *makkahCircle;
    IBOutlet UIButton *minaCircle;
    IBOutlet UIButton *arafatCircle;
    IBOutlet UIButton *muzdalifaCircle;

    IBOutlet UIImageView *nilArrow;
    IBOutlet UIImageView *miqatToMakkah;
    IBOutlet UIImageView *makkahToMina;
    IBOutlet UIImageView *minaToArafat;
    IBOutlet UIImageView *arafatToMuzdalifa;
    IBOutlet UIImageView *muzdalifaToMina;
    IBOutlet UIImageView *minaToMakkah;
}

@property int currentIndex;

@property(nonatomic, retain) NSArray *allArrows;
@property(nonatomic, retain) NSArray *allCircles;
@property(nonatomic, retain) NSArray *allDayLabels;
@property(nonatomic, retain) NSArray *allPositionLabels;

@property(nonatomic, retain) UILabel *dayLabel;
@property(nonatomic, retain) UILabel *positionLabel;

@property(nonatomic, retain) UIButton *miqatCircle;
@property(nonatomic, retain) UIButton *makkahCircle;
@property(nonatomic, retain) UIButton *minaCircle;
@property(nonatomic, retain) UIButton *arafatCircle;
@property(nonatomic, retain) UIButton *muzdalifaCircle;

@property(nonatomic, retain) UIImageView *nilArrow;
@property(nonatomic, retain) UIImageView *miqatToMakkah;
@property(nonatomic, retain) UIImageView *makkahToMina;
@property(nonatomic, retain) UIImageView *minaToArafat;
@property(nonatomic, retain) UIImageView *arafatToMuzdalifa;
@property(nonatomic, retain) UIImageView *muzdalifaToMina;
@property(nonatomic, retain) UIImageView *minaToMakkah;

@end


//
//  MapView.m
//  AlHajj
//
//  Created by Shahzeb Khan on 12/7/13.
//  Copyright (c) 2013 Shahzeb Khan. All rights reserved.
//

#import "MapView.h"

@interface MapView ()

@end

@implementation MapView

@synthesize currentIndex;
@synthesize allArrows, allCircles, allDayLabels, allPositionLabels;
@synthesize dayLabel, positionLabel;
@synthesize miqatCircle, makkahCircle, minaCircle, arafatCircle, muzdalifaCircle;
@synthesize nilArrow, miqatToMakkah, makkahToMina, minaToArafat, arafatToMuzdalifa, muzdalifaToMina, minaToMakkah;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}
```

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    [self initializeView];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (void)initializeView {

    [self collectAllArrows];
    [self hideAllArrows];

    [self collectAllCircles];
    [self hideAllCirclesExceptCurrent:self.miqatCircle];

    [self collectAllDayLabels];
    [self collectAllPositionLabels];

    [self showInitialDayLabel];
    [self showInitialPositionLabel];

    self.currentIndex = 0;
}

- (IBAction)seeDetails:(id)sender {

    [self performSegueWithIdentifier:@"SeeDetails" sender:sender];
}

// In a story board-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
    if ([segue.identifier isEqualToString:@"SeeDetails"]) {
        UINavigationController *navigationController = segue.destinationViewController;
        DetailView *detailView = [[navigationController viewControllers] objectAtIndex:0];

        NSString *content = [self getContentForCurrentIndex:self.currentIndex];
        detailView.content = content;
    }
}

- (void)collectAllArrows {

    self.allArrows = @[self.nilArrow, self.miqatToMakkah, self.makkahToMina, self.minaToArafat, self.arafatToMuzdalifa,
                       self.muzdalifaToMina, self.minaToMakkah, self.makkahToMina, self.minaToMakkah];
}

- (void)hideAllArrows {

    for (UIImageView *imageView in self.allArrows) {
        imageView.hidden = YES;
    }
}


- (void)collectAllCircles {

    self.miqatCircle.tag = 0;
    self.makkahCircle.tag = 1;
    self.minaCircle.tag = 2;
    self.arafatCircle.tag = 3;
    self.muzdalifaCircle.tag = 4;

    self.allCircles = @[self.miqatCircle, self.makkahCircle, self.minaCircle, self.arafatCircle, self.muzdalifaCircle,
                        self.minaCircle, self.makkahCircle, self.minaCircle, self.makkahCircle];
}

- (void)hideAllCirclesExceptCurrent:(UIButton *)currentCircle {

    for (UIButton *button in self.allCircles) {
        if (button.tag == currentCircle.tag) {
            button.hidden = NO;
        } else {
            button.hidden = YES;
        }
    }
}
```

```
- (void)collectAllDayLabels {

    NSString *label1 = @"Day 1, 7th of Dhu'l-Hajjah";
    NSString *label2 = @"Day 1, 7th of Dhu'l-Hajjah";
    NSString *label3 = @"Day 2, 8th of Dhu'l-Hajjah";
    NSString *label4 = @"Day 3, 9th of Dhu'l-Hajjah";
    NSString *label5 = @"Day 3, 9th of Dhu'l-Hajjah";
    NSString *label6 = @"Day 4, 10th of Dhu'l-Hajjah";
    NSString *label7 = @"Day 4, 10th of Dhu'l-Hajjah";
    NSString *label8 = @"Day 5, 11th of Dhu'l-Hajjah";
    NSString *label9 = @"Day 7, 13th of Dhu'l-Hajjah";

    self.allDayLabels = @[label1, label2, label3, label4, label5, label6, label7, label8, label9];

}

- (void)collectAllPositionLabels {

    NSString *label1 = @"You are at Miqat,";
    NSString *label2 = @"You are at Makkah,";
    NSString *label3 = @"You are at Mina,";
    NSString *label4 = @"You are at Arafat,";
    NSString *label5 = @"You are at Muzdalifah,";
    NSString *label6 = @"You are at Mina,";
    NSString *label7 = @"You are at Makkah,";
    NSString *label8 = @"You are at Mina,";
    NSString *label9 = @"You are at Makkah,";

    self.allPositionLabels = @[label1, label2, label3, label4, label5, label6, label7, label8, label9];
}

- (IBAction)resetMap:(id)sender {

    [self initializeView];
}

- (IBAction)goToNextStop:(id)sender{

    if (currentIndex > 7) {
        [self displayAlert];
        return;
    }

    [self showNextArrow];
    [self showNextCircle];
    [self showNextDayLabel];
    [self showNextPositionLabel];

    self.currentIndex++;
}

- (IBAction)goToPreviousStop:(id)sender{

    if (currentIndex == 0) { return; }

    [self hideCurrentArrow];
    [self showPreviousCircle];
    [self showPreviousDayLabel];
    [self showPreviousPositionLabel];

    self.currentIndex--;
}

- (void)showNextArrow {

    UIImageView *nextArrow = self.allArrows[self.currentIndex+1];
    nextArrow.hidden = NO;
}


- (void)hideCurrentArrow {

    UIImageView *currentArrow = self.allArrows[self.currentIndex];
    currentArrow.hidden = YES;
}

- (void)showNextCircle {

    UIButton *nextCircle = self.allCircles[self.currentIndex+1];
    nextCircle.hidden = NO;

    [self hideCurrentCircle];
}

- (void)showPreviousCircle {

    UIButton *previousCircle = self.allCircles[self.currentIndex-1];
    previousCircle.hidden = NO;

    [self hideCurrentCircle];
}

- (void)hideCurrentCircle {

    UIButton *currentCircle = self.allCircles[self.currentIndex];
    currentCircle.hidden = YES;
}
```

```
- (void)showInitialDayLabel {

    NSString *label = self.allDayLabels[0];
    self.dayLabel.text = label;
}

- (void)showNextDayLabel {

    NSString *label = self.allDayLabels[self.currentIndex+1];
    self.dayLabel.text = label;
}

- (void)showPreviousDayLabel {

    NSString *label = self.allDayLabels[self.currentIndex-1];
    self.dayLabel.text = label;
}

- (void)showNextPositionLabel {

    NSString *label = self.allPositionLabels[self.currentIndex+1];
    self.positionLabel.text = label;
}

- (void)showPreviousPositionLabel {

    NSString *label = self.allPositionLabels[self.currentIndex-1];
    self.positionLabel.text = label;
}

- (void)showInitialPositionLabel {

    NSString *label = self.allPositionLabels[0];
    self.positionLabel.text = label;
}

- (NSString *)getContentForCurrentIndex:(int)index {

    NSString* path = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"%d", index] ofType:@"txt"];
    return [NSString stringWithContentsOfFile:path encoding:NSUTF8StringEncoding error:NULL];
}

- (void)displayAlert {

    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Hajj Complete"
                                                    message:@"You have covered the entire Hajj procedure. Would you like to start over?"
                                                   delegate:self cancelButtonTitle:@"Cancel" otherButtonTitles:@"Yes", nil];

    [alert show];
}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex {

    if (buttonIndex == 1) {
        [self initializeView];
    }
}

@end
```

Fig. b2: Interactive Map Code.

```
//
//  ContactsView.h
//  AlHajj
//
//  Created by Shahzeb Khan on 11/3/13.
//  Copyright (c) 2013 Shahzeb Khan. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "ContactCell.h"
#import "ContactItem.h"
#import "ContactAddView.h"
#import "ContactAddViewDelegate.h"

@interface ContactsView : UITableViewController <UIAlertViewDelegate, ContactAddViewDelegate> {

    NSMutableArray *userContacts;
    NSArray *defaultContacts;
    NSString *currentNumber;
}

@property (nonatomic, retain) NSMutableArray *userContacts;
@property (nonatomic, retain) NSArray *defaultContacts;
@property (nonatomic, retain) NSString *currentNumber;

@end

//
//  ContactsView.m
//  AlHajj
//
//  Created by Shahzeb Khan on 11/3/13.
//  Copyright (c) 2013 Shahzeb Khan. All rights reserved.
//

#import "ContactsView.h"

@interface ContactsView ()

@end

@implementation ContactsView

@synthesize userContacts, defaultContacts, currentNumber;

#pragma mark - View lifecycle

- (id)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    // Uncomment the following line to preserve selection between presentations.
    // self.clearsSelectionOnViewWillAppear = NO;

    // Uncomment the following line to display an Edit button in the navigation bar for this view controller.
    // self.navigationItem.rightBarButtonItem = self.editButtonItem;
    self.userContacts = nil;
    self.userContacts = [[NSMutableArray alloc] init];

    [self readContacts];
    [self readUserContactlist];
}

- (void)viewWillAppear:(BOOL)animated {

    [self.tableView reloadData];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
```

```
#pragma mark - Alert view delegate methods

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex {

    if (buttonIndex == 1) {
        // Dial cached number
        [[UIApplication sharedApplication] openURL:[NSURL URLWithString:[NSString stringWithFormat:@"tel:%@", self.currentNumber]]];
    } else {
        // Empty cached number
        self.currentNumber = nil;
    }
}

#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Return the number of sections.
    return 2;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // Return the number of rows in the section.
    if (section == 0) {
        return [self.defaultContacts count];
    } else {
        NSLog(@"COUNT: %d", [self.userContacts count]);
        return [self.userContacts count];
    }
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"ContactCell";
    ContactCell *cell = (ContactCell *)[tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];

    if (cell == nil) {
        cell = [[ContactCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier];
    }
    // Configure the cell...
    NSString *name;
    NSString *number;

    if (indexPath.section == 0) {
        NSDictionary *dict = self.defaultContacts[indexPath.row];
        name = dict.allKeys[0];
        number = [dict valueForKey:name];
    } else {
        ContactItem *dict = self.userContacts[indexPath.row];
        name = dict.name;
        number = dict.number;
    }

    cell.name.text = name;
    cell.number.text = number;

    return cell;
}


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

    NSString *name;
    NSString *number;

    if (indexPath.section == 0) {
        NSDictionary *dict = self.defaultContacts[indexPath.row];
        name = dict.allKeys[0];
        number = [dict valueForKey:name];
    } else {
        ContactItem *dict = self.userContacts[indexPath.row];
        name = dict.name;
        number = dict.number;
    }

    // Cache the number into an instance variable to dial later
    self.currentNumber = number;

    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Call"
                                                    message:[NSString stringWithFormat:@"Are you sure you want to call %@?", number]
                                                   delegate:self
                                          cancelButtonTitle:@"Cancel"
                                          otherButtonTitles:@"Yes", nil];
    [alert show];
}

// Override to support editing the table view.
- (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        // Delete the row from the data source
        [self.userContacts removeObjectAtIndex:indexPath.row];
        [self writeContactsToPlist];

        [tableView deleteRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationFade];
    }
    else if (editingStyle == UITableViewCellEditingStyleInsert) {
        // Create a new instance of the appropriate class, insert it into the array, and add a new row to the table view
    }
}
```

```
- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    if (indexPath.section == 0) {
        return NO;
    } else {
        return YES;
    }
}

- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {

    if (section == 0) {
        return @"Saudi Emergency Contacts";
    } else {
        return @"Your Emergency Contacts";
    }
}

#pragma mark - Navigation

// In a story board-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
    if ([segue.identifier isEqualToString:@"AddContact"]) {
        UINavigationController *navigationController = segue.destinationViewController;
        ContactAddView *contactAddView = [[navigationController viewControllers] objectAtIndex:0];
        contactAddView.delegate = self;
    }
}

#pragma mark - Checklist add view delegate methods

- (void)contactAddViewDidCancel {

    [self dismissViewControllerAnimated:YES completion:nil];
}

- (void)contactAddViewControllerDidSaveName:(NSString *)name andNumber:(NSString *)number {

    ContactItem *item = [[ContactItem alloc] initWithName:name andNumber:number];
    [self.userContacts addObject:item];

    [self writeContactsToPlist];

    [self dismissViewControllerAnimated:YES completion:nil];
}

#pragma mark - Read / write checklist

- (void)readUserContactlist {

    [self readUserContactsAtPath:[self pathForUserContactlist]];
}


-(void)readUserContactsAtPath:(NSString *)path {

    NSArray *array = [NSArray arrayWithContentsOfFile:path] ;

    for (NSDictionary *dict in array) {
        for (id key in dict) {
            NSString *name = key;
            NSString *number = [dict objectForKey:key];
            NSLog(@"NAME AND NUMBER: %@, %@", name, number);
            ContactItem *item = [[ContactItem alloc] initWithName:name andNumber:number];
            [self.userContacts addObject:item];
        }
    }
}

- (void)writeContactsToPlist {

    NSMutableArray *array = [[NSMutableArray alloc] init];

    for (ContactItem *item in self.userContacts) {
        NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
        [dict setValue:item.number forKey:item.name];
        [array addObject:dict];
    }

    [array writeToFile:[self pathForUserContactlist] atomically:YES];
}

- (NSString *)pathForUserContactlist {

    NSString *documentsDirectory = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) firstObject];
    return [NSString stringWithFormat:@"%@/Contacts.plist", documentsDirectory];
}

- (void)readContacts {

    self.defaultContacts = [NSArray arrayWithContentsOfFile:[self pathForContactlist]];
}

- (NSString *)pathForContactlist {

    return [[NSBundle mainBundle] pathForResource:@"Contacts" ofType:@"plist"];
}


@end
```

Fig. b3: Interactive Contact List Default.

```
//
//  ChirpsView.h
//  AlHajj
//
//  Created by Shahzeb Khan on 11/12/13.
//  Copyright (c) 2013 Shahzeb Khan. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ChirpsView : UIViewController {

    NSArray *chirps;
    IBOutlet UIDatePicker *picker;
    IBOutlet UISwitch *agree;
}

@property(nonatomic, retain) NSArray *chirps;
@property(nonatomic, retain) UIDatePicker *picker;
@property(nonatomic, retain) UISwitch *agree;

@end


//
//  ChirpsView.m
//  AlHajj
//
//  Created by Shahzeb Khan on 11/12/13.
//  Copyright (c) 2013 Shahzeb Khan. All rights reserved.
//

#import "ChirpsView.h"

@interface ChirpsView ()

@end

@implementation ChirpsView

@synthesize picker, agree, chirps;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


- (void)viewWillDisappear:(BOOL)animated {

    NSLog(@"DAY#: %d", [self getDayOfYear]);

    [self loadChirps];
    [self scheduleChirps];
}

- (void)viewWillAppear:(BOOL)animated {

    [self populateScheduledChirpsTimeInPicker];
    [self populateAgreeState];
}

- (void)populateScheduledChirpsTimeInPicker {

    NSDate *savedDate = [[NSUserDefaults standardUserDefaults] valueForKey:@"scheduledChirpsTime"];
    NSLog(@"DATE: %@", savedDate);
    if (savedDate) {
        self.picker.date = savedDate;
    } else {
        self.picker.date = [NSDate date];
    }
}
```

```
- (void)scheduleChirps {

    // Cancel all local notifications
    [[UIApplication sharedApplication] cancelAllLocalNotifications];

    // Save state of the agree switch
    [[NSUserDefaults standardUserDefaults] setBool:self.agree.on forKey:@"agree"];

    // Remove scheduled chirps time if agree switch is off
    if (!self.agree.on) {
        [[NSUserDefaults standardUserDefaults] setObject:nil forKey:@"scheduledChirpsTime"];
        return;
    }

    // Build date components
    NSCalendar *calendar = [[NSCalendar alloc] initWithCalendarIdentifier:NSGregorianCalendar];
    NSDateComponents *components = [[NSCalendar currentCalendar] components: NSCalendarUnitMinute | NSCalendarUnitHour |
                                    NSCalendarUnitDay | NSCalendarUnitMonth | NSCalendarUnitYear fromDate:self.picker.date];
    [components setSecond:00];

    NSDate *date = [calendar dateFromComponents:components];

    // Build day component for incrementing date
    NSDateComponents *dayComponent = [[NSDateComponents alloc] init];
    dayComponent.day = 1;

    // Build local notification
    UILocalNotification *localNotif = [[UILocalNotification alloc] init];

    int dayOfYear = [self getDayOfYear];

    for (int i=0; i<=60; i++) {
        localNotif.fireDate = date;
        localNotif.alertBody = self.chirps[dayOfYear];
        localNotif.alertAction = @"Action";

        NSLog(@"ALERT BODY: %@", localNotif.alertBody);

        // Schedule local notification
        [[UIApplication sharedApplication] scheduleLocalNotification:localNotif];

        // Increment date by one calendar day
        date = [calendar dateByAddingComponents:dayComponent toDate:date options:0];

        // Increment day of year
        if (dayOfYear == 367) {
            dayOfYear = 0;
        } else {
            dayOfYear++;
        }

    }

    // Save scheduled chirps time
    [[NSUserDefaults standardUserDefaults] setObject:localNotif.fireDate forKey:@"scheduledChirpsTime"];

    NSLog(@"NOTIF: %@", [[UIApplication sharedApplication] scheduledLocalNotifications]);
}

- (NSUInteger)getDayOfYear {

    NSCalendar *gregorian = [[NSCalendar alloc] initWithCalendarIdentifier:NSGregorianCalendar];
    NSUInteger dayOfYear = [gregorian ordinalityOfUnit:NSDayCalendarUnit inUnit:NSYearCalendarUnit forDate:[NSDate date]];

    return dayOfYear;
}


- (void)loadChirps {

    self.chirps = [NSArray arrayWithContentsOfFile:[self getChirpsPlistPath]];
}

- (NSString *)getChirpsPlistPath {

    return [[NSBundle mainBundle] pathForResource:@"Chirps" ofType:@"plist"];
}

@end
```

Fig. b4: Motivational Messages Code.