ACCOMODATION SEARCH APPLICATION ON iPHONE

PISUTH KASINPHILA

WARALAK SATTAYAPONG

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

IN COMPUTER SCIENCE (INTERNATIONAL PROGRAM)

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2011

Thesis Title	Accommodation Search Application on iPhone
Students	Mr. Pisuth Kasinphila Student ID: 51051186
	Ms. Waralak Sattayapong Student ID: 51051192
Degree	Bachelor of Science
Program	Computer Science (International Program)
Year	2011
Thesis Advisor	Dr.Rungrat Wiangsripanawan

ABSTRACT

This special project aim is to develop an iPhone application called "Home Finder". The main objective of the application is to use GPS to search for the available accommodations (for sale) which are located around the reference point. The application provides three different searching types: Search by Nearby, Search by Area and Search by Train. The result including the basic information of each accommodation is displayed on the screen in two different views: the list view and the map view. The application also provides a map guide to navigate the user to his selected accommodation and a call function allowing the user to make a phone call to the sale office directly. For the implementation part, the accommodation information is retrieved from the *Home buyer guide* company's web service using soap messages.

SQLite is used for the database on the phone. The Google Map API is used for the map function.

Keywords: iPhone, Application, House Finder, Accommodation, Xcode, LBS,SQLite

ACKNOWLEDGEMENTS

This special project Accommodation Search Application on iPhone would not have been possible without the help from many people.

Special thanks to our parents for their unconditional loves and supports especially Pisuth's mother who motivated us, helped us and gave us a lot of useful advices on how to develop a program on iPhone.

Special thanks to *Home Buyer Guide* Company for their supports and suggestions on the system design and data source that we used in this special project. Thanks for including us as one of their iOS developer team members.

Special thanks to Dr.Rungrat Wiangsripanawan for her advices and comments to solve problems that we had in this special project.

Special thanks to Asst.Prof.Dr.Korakot Prachumrak and Dr.Warangkhana Kimpan for their suggestions on the improvement and suggestions on the interface design of our application.

Last but not least, special thanks to all our friends for their warmth support and help throughout this special project.

Pisuth Kasinphila

Waralak Sattayapong

Contents

Contents (Cont.)

Page

Contents (Cont.)

Contents (Cont.)

Contents (Cont.)

List of Tables

Table **Page**

List of Figures

Figure **Page**

List of Figures (Cont.)

Figure **Page**

List of Figures (Cont.)

List of Figures (Cont.)

Figure **Page**

List of Figures (Cont.)

List of Figures (Cont.)

Figure **Page**

Chapter 1

Introduction

1.1 Rational and Research motivation

Nowadays mobile phones are not only used for communication with others. Furthermore, in the age of communication (3G), iPhone is one of the most popular smart phones. It can access to many source of information; people use iPhone to access to all information available on the Internet. Therefore, we have a motivation to develop an iPhone application to search for houses or condominiums that are for sale around the phones location. This application will help its user to search for houses or condominiums. To illustrate, the application allows the user to search houses in 2 kilometers distance from his position. When the user uses this function, all the available houses and condominiums will be displayed with the specific symbols on the map. This map together with some additional information will help the user to make decision easier. Example of additional information are the area that the accommodation is located, accommodation price, the distance to the public transportation such as BTS or MRT. In addition, the application will provide the user a route to navigate to his interested accommodation including the call function which allows him to call the sale office directly from the application.

1.2 Purposes of the special project

1. This application enables a user to search for a house or a condominium within the specified distance using his iPhone.
2. This application will decrease time to plan a route to the selected accommodation using a map on this application.
3. This application will decrease the driving time when a user has more than one-interested houses or condominiums in the same route or area.
4. This application will provide the call function, which allows the user to make a phone call to the selected accommodation directly.

1.3 Coverage of the special project

1. Study how to develop an iPhone application using Xcode and Cocoa
2. Study how to develop an application using Objective-C.
3. Study the web service to get data from website (database).
4. Study how the Google Map API works.
5. Study SQLite.

1.4 Expected benefits

For Developer

- Gain knowledge in Objective-C language to develop an application on iOS device using Xcode.

- Gain knowledge to manage the database in SQLite.
- Gain knowledge to use Google Map API to calculate distance from latitude and longitude and method to show map of house.

For User

- Know the available accommodations around the selected location.
- Know about the information of the selected house.
- Help the user to make a decision for choosing the most suitable accommodation for him.

1.5 Implementation procedures

1. Understand the requirements of this project.
2. Create a project plan and draw a Gantt chart of the project.
3. Choose tools for developing the application and study how to use these tools.
4. Separate the application into modules and do the programming for each module.
5. Integrate each module together for a complete application.
6. Test the whole application and fix all errors or bugs.

1.6 Organization

This special project consists of 5 chapters.

Chapter 1: Introduction

Chapter 2: Background

Chapter 3: System Analysis and Design

Chapter 4: Implement of this special project

Chapter 5: Conclusion of this special project

1.7 Equipments used to implement this special project

1.7.1 Hardware

1. One MacBook Pro
   - Hard disk 320 GB
   - Ram 4 GB
   - CPU core i5
2. One iPhone 4S

1.7.2 Software

- Mac OS X 10.6
- Xcode 4.1
- SQLite
- Google Maps API

Chapter 2

Background

2.1 iOS [1]

iOS known as iPhone OS is Apple's mobile operating system. Originally developed for the iPhone, it has since been extended to support other Apple, Inc. devices such as the iPod touch, iPad and Apple TV. Apple, Inc. does not license iOS for installation on third-party hardware.

The user interface of iOS is based on the concept of direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons. The response to user input is immediate. Interaction with the OS includes gestures such as swipe, tap, pinch, and reverse pinch, all of which have specific definitions within the context of the iOS operating system and its multitouch interface. Internal accelerometers are used by some applications to respond to shaking the device (one common result is the undo command) or rotating it in three dimensions (one common result is switching from portrait to landscape mode).

iOS is derived from Mac OS X. There are four abstraction layers: the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer. The current version of the operating system (iOS 4.3.5) uses roughly 650 megabytes of the device's storage, varying for each model.

The iPod touch retains the same applications that are present by default on the iPhone, with the exception of the Phone, Messages, Compass and Camera apps. The "iPod"

App presented on the iPhone is split into two apps on the iPod Touch: Music, and Videos. The bottom row of applications is also used to delineate the iPod touch's main purposes: Music, Videos, Safari, and App Store (Dock Layout was changed in 3.1 Update). For the 4th Generation iPod touch, it includes FaceTime and Camera, and the dock layout had changed to Music, Mail, Safari, and Video. As of iOS 5.0 (to be released to the public in September to November 2011), "iMessage" will be available on all iOS devices running iOS 5. iMessage is effectively a version of the iPhone Messages app that sends free text or multimedia messages to other iOS devices (similar to BlackBerry Messenger).

The iPad comes with the same applications as the iPod touch excluding Stocks, Weather, Clock, Calculator, and the Nike + iPod app. Separate music and video apps are provided, as on the iPod touch, although (as on the iPhone) the music app is named "iPod". In iOS 5, the iPod app will be replaced by Music and Video apps on all devices. Most of the default applications are completely rewritten to take advantage of the iPad's larger display. The default dock layout includes Safari, Mail, Photos and iPod.

2.1.1 Multitasking

Before iOS 4, multitasking was limited to a selection of the applications Apple included on the devices. Apple worried that running multiple third-party applications simultaneously would drain batteries too quickly. Starting with iOS 4, on 3rd-generation and newer iOS devices, multitasking is supported through seven background APIs:

- Background audio

- Voice over IP
- Background location
- Push notifications
- Local notifications
- Task finishing
- Fast app switching

Developing iOS application is done using Xcode[1], the integrated development environment (IDE) used to develop iOS applications. With Xcode, developers can organize and edit their source files, view documentation, build your application, debug your code, and optimize your application's performance.

The iOS application development process is classified into the following major steps:

1. **Create your project**: Xcode provides several project templates so that developers can choose the template that implements the type of application that they want.
2. **Design the user interface**: The Interface Builder application allows developers to design their application's user interfaces graphically in which can be saved as resource files that their application can load at runtime. In case that developers do not want to use the Interface Builder, they can layout their user interface programmatically.
3. **Write code**: Xcode provides several features that allow its developers to write code fast. They are class and data modeling, code completion, direct access to

documentation, and refactoring.

4. **Build and run application**: Developers build their applications on their computers and can run these applications in either the iOS simulation environment or on their devices.

2.1.2 Component of the iPhone SDK

The iPhone SDK includes a suite of development tools to help the developers to create applications for iPhone, iPod touch, and iPad. It consists of the following components:

- Xcode

  Xcode is an Integrated development environment (IDE) that enables developers to manage, edit, and debug their projects.

- Dashcode

  Integrated development environment (IDE) that enables developers to develop web based iPhone and iPad application and Dashboard Widgets.

- iPhone Simulator

  It provides a software simulator to simulate an iPhone or an iPad on apple computer (Mac).

- Interface Builder

  A visual editor for designing iPhone and iPad application's user interfaces.

- Instruments

  An analysis tool for helping developers both to optimize their applications and monitor for memory leaks in real time.

2.2 Xcode [2]

Xcode is a suite of tools, developed by Apple, for developing software for Mac OSX and iOS. The main application of the suite is the integrated development environment (IDE), also named Xcode. The Xcode suite also includes most of Apple's developer documentation, and Interface Builder, an application used to construct graphical user interfaces.

The Xcode suite includes a modified version of free software GNU Compiler Collection. The Xcode developer tools package provides everything Mac and iOS developers need for creating the applications.

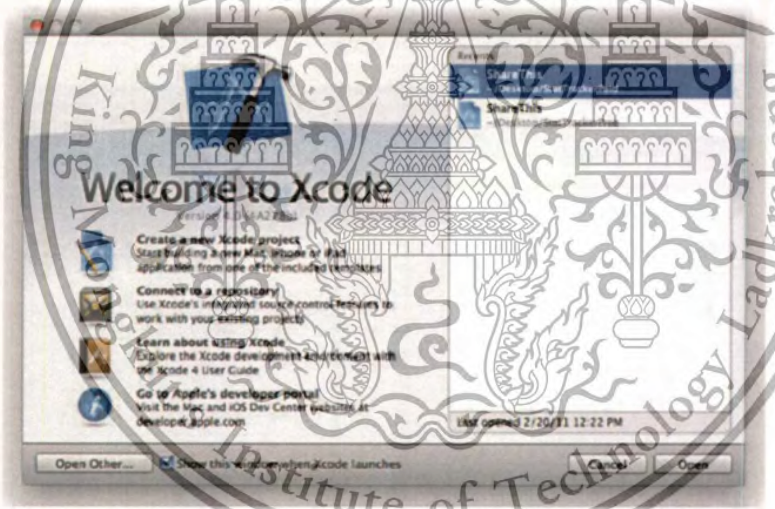


Figure 2.1 Xcode4 Startup window

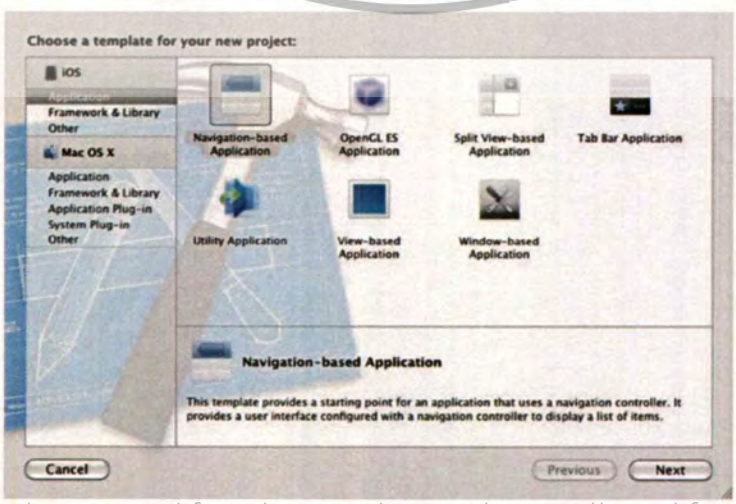


Figure 2.2 Project Template, which Xcode provide for user

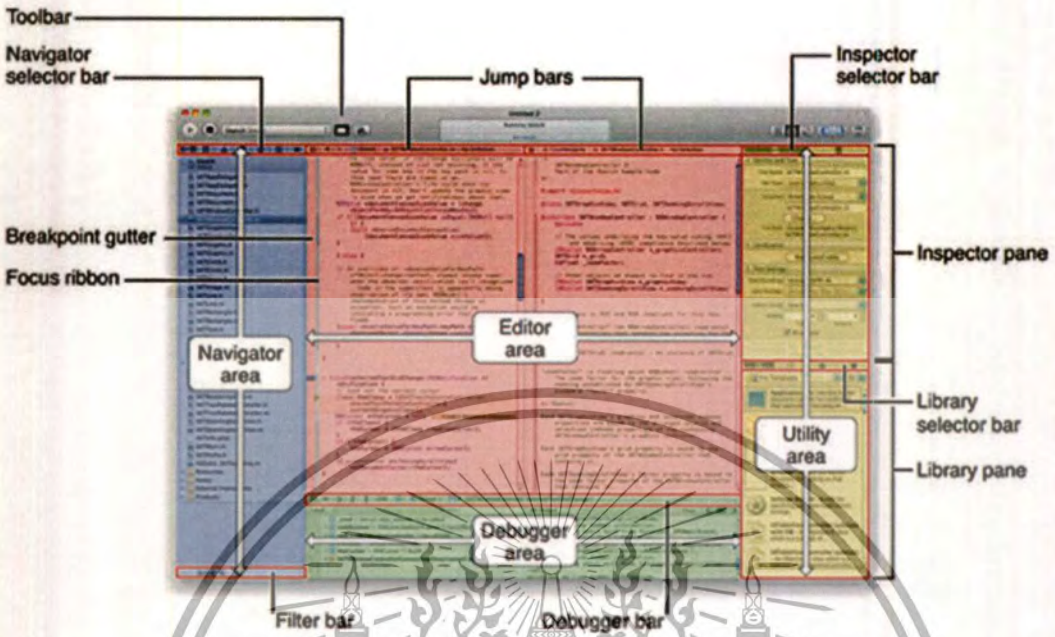


Figure 2.3 Xcode4 GUI

Xcode is tightly integrated with the Cocoa and Cocoa Touch frameworks [1].The Xcode toolset includes the Xcode IDE, with the Interface Builder design tool and Apple LLVM compiler fully integrated. The Instruments analysis tool is also included together with several other supporting developer tools

2.2.1 Xcode IDE

Xcode integrates all the tools. The unified interface provides all interfaces from composing source code, to debugging, and even to designing the next user interface, all within the same window. Live Issues will immediately alert developers when the coding mistakes are typed by displaying a message bubble beside the code for more detail. Hit the Run button is also provided for launching the Mac app, or for

uploading the app to developer's test device, and immediately starts debugging. To inspect the variable values at runtime can be simply done by moving the mouse to point on those variables so that the developer does not have to lose his place in the editor. By working closely with the developer web portal, Xcode discovers new iOS devices with a single click; it also allow developers to securely sign and archive their Mac or iOS apps including directly submit them to the App Store.

2.2.2 Apple LLVM Compiler [3]

Apple LLVM (Low Level Virtual Machine) is the next-generation compiler technology used in Xcode 4. It is developed by Apple engineers to be used specifically with iPhone, iPad, and the multi-core Mac. It is claimed by [reference] that it compiles code two times faster than GCC so that it produces applications that also run faster. The full Apple LLVM compiler stack — from the front-end parser, to the back end code optimizer —in Xcode 4 has a great support for C, Objective-C, and C++.

Syntax highlighting, code completion, and every other index-driven feature is handled by the LLVM parser. If the compiler knows about a symbol, so does the Xcode IDE. C, C++, and Objective-C are all accurately understood at editing time, exactly as they are when building.

2.2.3 Fix-it and Live Issues

The Apple LLVM engine is constantly working in the background to understand developers' codes. In the editor, the new Live Issues feature uses that understanding to alert developers to coding mistakes as they type. Similar to a word

processor highlighting spelling errors, Xcode 4 highlights common coding mistakes, without the need to click 'build' first.

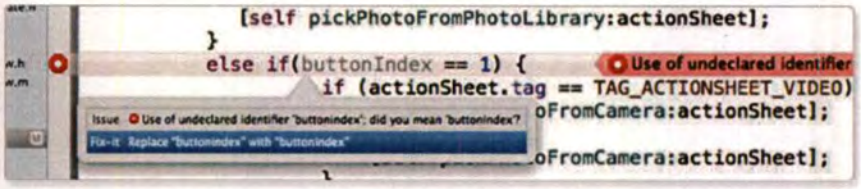


Figure 2.4 Fix-it in Xcode4

The IDE is very intelligent. I can fix the problem for coding error. In many cases, Xcode will not only report an error but it will also present a solution. This can be done by clicking at the error to see the available Fix-its, such as correcting an assignment to a comparison, repairing a misspelled symbol, or appending a missing semicolon. A single keyboard shortcut will instantly repair the error so that the programmer can continue coding.

Fix-it is comes with the Analyze feature. The Xcode static analyzer checks through thousands of potential code paths, looking for places where code, while valid, would behave in unexpected ways, such as memory allocation mistakes, never-hit case statements, or improperly constructed loops.

2.2.4 Version Editor

The new Version editor in Xcode 4 makes it easy to see any two versions of source code, side by side, in a live editor. It is a new way to think of source control management in an IDE. The comparison view is also a timeline. Drag the slider in the middle and the developers go back in time through their projects, comparing any two versions.

Figure 2.5 Version editor in Xcode4

The Version editor can also show a detailed log of past events, and track blame for past check-ins. Complex SCM (software configuration management) commands are managed behind the scenes. It is possible to manage multiple projects within a single Xcode 4 workspace. That is one project managed in Subversion, the other in Git, all updated automatically.

2.2.5 Debugger

Xcode 4 introduces LLDB (Low Level Debugger) [4], a brand new debugging engine contributed by Apple to the LLVM.org open source project. Like LLVM, the new LLDB engine is designed from the ground up to consume much less memory, and fast.

When the app is running, the navigator will show a stack trace that can be expanded or compressed to show or hide stack frames as programmers debug. As step through, programmers **can even lock onto a single thread then click "continue" and follow that**

specific thread of execution. Multicore debugging in Xcode 4 is now as easy as multicore coding with blocks and Grand Central Dispatch.

2.2.6 iOS Simulator [5]

The iOS Simulator runs the application in the same way as an actual iOS device. Because it is quick to launch and debug, the iOS Simulator is used for the test bed to make sure that the app's user interface works as designed, your network calls are correct, and that the views changed correctly when the phone rotates. Developers can simulate touch gestures by using the mouse.

Figure 2.6 Xcode4 iOS simulator

2.2.7 Interface Builder

In Xcode 4, Interface Builder has been completely integrated within the Xcode IDE Selecting an interface file (*.nib/.xib*) in the project will open the IB editor within Xcode. Opening the right-hand Utility area will show the full complement of interface inspectors, as well as the library of controls and UI objects. Drag a control from the

Figure 2.7 Xcode4 Interface Builders

library, and drop onto the canvas, to layout the Mac OS X or iOS application.

Developers are allowed to drag connections directly from the UI design to the source code. If the codes are not ready to connect Xcode will create a new outlet or action; developers just drag to an empty space in their source files and Xcode will generate the code.

Figure 2.8 Creating new action by Xcode

2.2.8 Assistant

Whether writing code, or designing an interface, no change happens in isolation. Even the smallest edit can have a cascading effect throughout the project. Developers often need to see more than just your current file; they need to see related documents as well.

Figure 2.9 Xcode assistant

With the Xcode Assistant, the two-pane editor is used. When Assistant is turned on, the IDE will anticipate which other files developers need to see. For example, if it is editing a new derived class, the Assistant will show the code for the class that is inherited; if it is writing a new implementation code, the assistant will automatically show the corresponding header. When designing an interface, the Assistant will show

the appropriate controller, making drag-and-drop code connections extremely simple. Data model designing will bring up the classes that back developers' models — all automatically.

2.2.9 Instruments for Xcode

Instruments have streamlined interface, completed with the new Jump Bar, and stack compression from Xcode 4's UI. Using Xcode4's new launch schemes, it is easier to create a robust test to run app in Instruments. That includes launching Instruments in "deferred mode", keeping the data collection UI off screen and saving system resources for running and metering the application.

Figure 2.10 Xcode Instruments

New data collection instruments are also available, including OpenGL ES for tracking iPhone graphics performance, new memory allocation monitoring that can find unintended memory growth, Time Profiler on iOS for collecting samples with very low overhead, and complete System Trace for checking how all system processes interact.

2.3 Google Maps API [5]

The information in this section is extracted from [5].

Google Maps (Google Local) is a web mapping service application and technology provided by Google, for non-commercial use. It provides many map-based services, including the Google Maps website, Google Ride Finder, Google Transit, and maps embedded on third-party websites via the Google Maps API. It offers street maps, a route planner for traveling by foot, car, bike (beta) or public transport and an urban business locator for numerous countries around the world. Google Maps satellite images are not updated in real time; they are several months or years old.

Google Maps uses a close variant of the Mercator projection, so it cannot show areas around the poles. A related product is Google Earth, a stand-alone program which offers more global - viewing features, including showing polar areas.

2.3.1 Satellite view

The information in this subsection is extracted from [5].

Google Maps provides high-resolution aerial or satellite images for most urban areas in the United States (including Hawaii, Alaska, Puerto Rico, and the U.S. Virgin Islands), Canada, and the United Kingdom, as well as parts of Australia and many other countries. The high-resolution imagery has been used by Google Maps to cover all of Egypt's Nile Valley, Sahara desert and Sinai. Google Maps also covers many cities in the English speaking areas. However, Google Maps is not solely an English maps service, since its service is intended to cover the world. Various governments

have complained about the potential for terrorists to use the satellite images in planning attacks. Google has blurred some areas for security (mostly in the United States), including the U.S. Naval Observatory area (where the official residence of the Vice President is located), and previously the United States Capitol and the White House. Other well-known government installations, including Area 51 in the Nevada desert, are visible. Not all areas on satellite images are covered in the same resolution; less populated areas usually get less detail. Patches of clouds may obscure some areas. With the introduction of an easily plan and searchable mapping and satellite imagery tool, Google's mapping engine prompted a surge of interest in satellite imagery. Sites were established which feature satellite images of interesting natural and man-made landmarks, including such novelties as "large type" writing visible in the imagery, as well as famous stadia and unique geological formations. Although Google uses the word satellite, most of the high-resolution imagery of cities is aerial photography taken from aircraft flying at 800–1500 feet rather than from satellites; while most of the rest of the imagery is in fact from satellites.

2.3.2 Direction

The information in this subsection is extracted from [5]

The "Get Directions" option provides a route planner. Depending on the area, it may allow the choice between "by car", "by public transit", "walking", and "bicycling".

Google Maps directions works for the option "by car":

Figure 2.11 Route planner work with Google Maps API to get direction

- Contiguously in Europe, the Middle East, South Asia and the African Mainland
- Contiguously in North America: Belize, Canada, Mexico, and the United States
- Contiguously in South America: Argentina, Bolivia, Brazil, Chile, Paraguay and Peru (most parts, except e.g. Iquitos and other places in the Loreto region)
- Contiguously in Southeast Asia: Cambodia, Laos, Singapore, peninsular Malaysia,

Thailand and Vietnam

- In certain noncontiguous countries and regions

2.3.3 Implementation

Like many other Google web applications, Google Maps uses JavaScript extensively [6]. As the user drags the map, the grid squares are downloaded from the server and inserted into the page. When a user searches for a business, the results are downloaded in the background for insertion into the side panel and map; the page is not reloaded. Locations are drawn dynamically by positioning a red pin (composed of several partially-transparent PNGs) on top of the map images.

2.3.4 Extensibility and customization

As Google Maps is coded almost entirely in JavaScript and XML, some end users have reverse-engineered the tool and produced client-side scripts and server-side hooks which allowed a user or website to introduce expanded or customized features into the Google Maps interface.

Using the core engine and the map/satellite images hosted by Google, such tools can introduce custom location icons, location coordinates and metadata, and even custom map image sources into the Google Maps interface. The script-insertion tool *Greasemonkey* [7] provides a large number of client-side scripts to customize Google Maps data.

Combinations with photo sharing websites, such as Flickr, are used to create "memory maps". Using copies of the Keyhole satellite photos, users have taken advantage of image annotation features to provide personal histories and information regarding particular points of the area [8].

2.3.5 Google Maps API on Website

The information in this section is extracted from [9].

After the success of reverse-engineered mash up such as chicagocrime.org and housingmaps.com, Google launched the Google Maps API in June 2005 to allow developers to integrate Google Maps into their websites. It is a free service, and currently does not contain ads, but Google states in their terms of use that they reserve the right to display ads in the future.

By using the Google Maps API, it is possible to embed Google Maps site into an external website, on to which site specific data can be overlaid. Although initially only a JavaScript API, the Maps API has since expanded to include an API for Adobe Flash applications, a service for retrieving static map images, and web services for performing geocoding, generating driving directions, and obtaining elevation profiles. Over 350,000 web sites use the Google Maps API, making it the most heavily used web application development API.

The Google Maps API is free for commercial use providing that the site on which it is being used is publicly accessible and does not charge for access. Sites that do not meet these requirements can purchase Google Maps API Premier.

The success of the Google Maps API has spawned a number of competing alternatives, including the Yahoo! Maps API [10], Bing Maps Platform [11], MapQuest Development Platform [12], and OpenLayers [13].

Figure 2.12 Google Maps API on Google website

2.3.6 Google Maps for Mobile

The information in this subsection is extracted from [5].

Google introduced a Java application called Google Maps for Mobile, intended to run on any Java-based phone or mobile device. Many of the web-based site's features are provided in the application.

Google Maps for Mobile 2.0 introduced a GPS-like location service that does not require a GPS receiver. The "my location" feature works by utilizing the GPS location of the mobile device, if it is available. This information is supplemented by the

software determining the nearest wireless networks and cell sites. The software then looks up the location of the cell

The Cell-site location method is used by triangulating the different signal strengths from different cell transmitters and then using their location property (retrieved from the online cell site database) to aid My Location in determining the user's current location. Wireless network location method is calculated by discovering the nearby Wi-Fi hotspots and using their location property (retrieved from the online Wi-Fi database, in the same way as the cell site database) to further discover the user's location.

The order in which these take precedence is:

- GPS-based services
- WLAN - based services, Wi-Fi based services
- Cell transmitter-based services

The software plots the streets in blue that are available with a yellow icon and a green circle around the estimated range of the cell site based on the transmitter's rated power (among other variables). The estimate is refined using the strength of the cell phone signal to estimate how close to the cell site the mobile device is.

2.4 Location Based Services [14]

Mobile phones and the Internet have revolutionized the communication and with it the lifestyle of people. An increasing number of mobile phones and Personal Digital Assistants (PDA) allow people to access the Internet whereever they are and whenever they want. From the Internet they can obtain on one hand information on events (cinema, concerts, parties) and on the other hand information on places (city maps, restaurants, museums, hospitals).

LBSs are information services accessible with mobile devices through the mobile network and utilizing the ability to make use of the location of the mobile device. LBS give the possibility of a two-way communication and interaction. Therefore the user tells the service provider his actual context like the kind of information he needs, his preferences and his position. This helps the provider of such location services to deliver information tailored to the user needs.

2.4.1 What are Location Based Services?

The information in this subsection is extracted from [14].

In the following subsections some major characteristics and definitions on LBS will be given. We will discuss the relation between GIS and LBS and give some Keywords, which are useful to describe the LBS Technology. Later the basic LBS components are introduced shortly. Finally we will explain what a Push and a Pull Service is.

2.4.2 The relation of GIS and LBS

The information in this subsection is extracted from [14].

Figure 2-14 shows that GIS and LBS have some particular similarities. Such common features are the handling of data with positional reference and spatial analysis functions. But LBS and GIS have different origins and different user groups as described by Virrantaus et al [21]. They analyze that Geographic Information

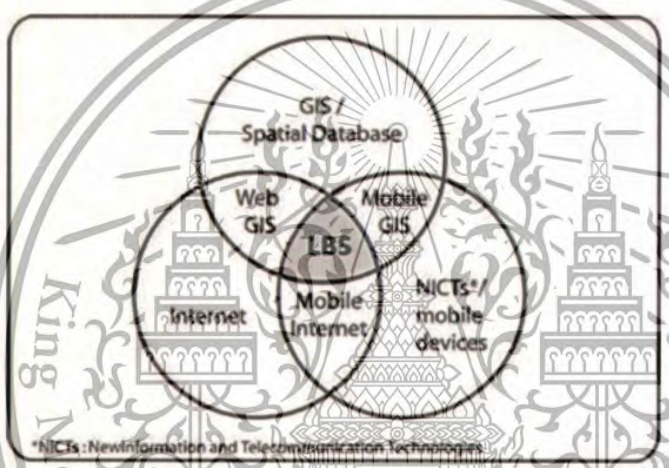


Figure 2.13 LBS as an intersection of technologies

Systems have been developed during several decades on the basis of professional geographic data applications whereas LBS were born quite recently by the evolution of public mobile services. With respect to user groups, GIS can be seen as traditional "professional" systems intended for experienced users with wide collection of functionality. Furthermore GISystems require extensive computing resources. In contrast, the LBS are developed as limited services for large non-professional user groups. Such LBS applications operating with the restrictions of mobile computing environment like low computational power, small displays or battery run time of the mobile device.

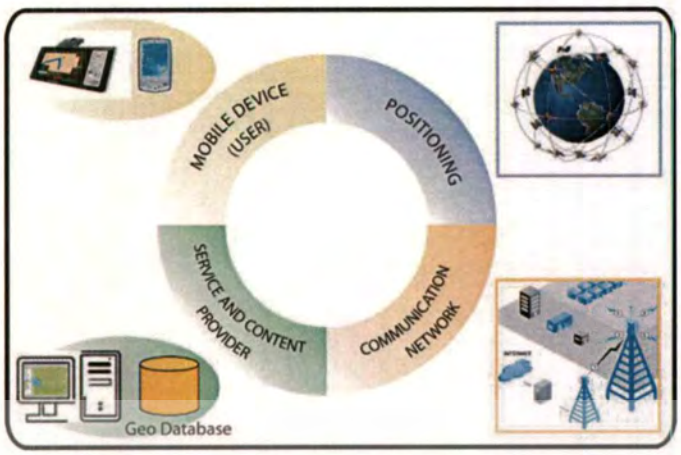


Figure 2.14 The basic components of LBS: User, Communication Network, Positioning, Service Provider and Content Provider.

2.4.3 LBS Components

The information in this subsection is extracted from [14].

If the user wants to use a Location Based Service different infrastructure elements are necessary. In Figure 2-15 the five (4+1) basic components and their connections are shown:

1. **Mobile Devices**: A tool for the user to request the needed information. The results can be given by speech, using pictures, text and so on. Possible devices are PDA's, Mobile Phones, Laptops, but the device can also be a navigation unit of car or a toll box for road pricing in a truck.
2. **Communication Network**: The second component is the mobile network which transfers the user data and service request from the mobile terminal to the service provider and then the requested information back to the user.
3. **Positioning Component**: For the processing of a service usually the user position has to be determined. The user position can be obtained either by using the mobile communication network or by using the Global Positioning

System (GPS). Further possibilities to determine the position are WLAN stations, active badges or radio beacons. The latter positioning methods can especially used for indoor navigation like in a museum. If the position is not determined automatically, the user also can specify it manually.

4. **Service and Application Provider**: The service provider offers a number of different services to the user and is responsible for the service request processing. Such services offer the calculation of the position, finding a route, searching yellow pages with respect to position or searching specific information on objects of user interest (e.g. a bird in wild life park) and so forth.
5. **Data and Content Provider**: Service providers will usually not store and maintain all the information, which can be requested by users. Therefore geographic base data and location information data will be usually requested from the maintaining authority or business and industry partners.

2.4.4 User Actions and Goals

The information in this subsection is extracted from [14].

The most obvious question is to know where the user himself is with respect to somebody or something else (locating). Users may search for persons, objects or events (searching) and they ask for the way to a location (navigating). Other questions ask for properties of a location (identifying) or they would try to look for events at or nearby a certain location (checking). It should be noted that checking uses not only geo-information but involves also time, since it refers to state of entities or events as well.

2.4.5 Information for searching, identifying and checking

The information in this subsection is extracted from [14].

The two basic actions locating and navigating mainly rely on geospatial information. Searching, identifying and checking however need a bigger variety of different information. Additionally to the geospatial information also other types of information are needed:

- Comprehensive static information is mainly contents such as yellow pages. Such information stays constant over a while and could of course also be retrieved via other media (book, newspaper, map, TV, internet, etc.).
- Topical information that may change while the user is on the move. In such a case the information checked previously from other media may no longer be valid. Examples of such topical information are traffic information, weather forecasts, last-minute theatre ticket deals, or online chat. In addition to topical information, the users will need guidance on how to proceed in the changed situation. For instance, a train schedule as such can be obtained elsewhere but once on the move, the user will need information on delays and estimated arrival times.
- Additionally safety information has key importance, e.g. actual information on the state of the roads or hiking trials, weather changes, danger of falling rocks, etc. Car drivers or boaters also need information in emergency situations, e.g. roadside help in a situation when the car breaks down.
- Far too often users are seen as passive information consumers. However, letting the users participate and provide their opinions and recommendations

could enhance many services with personal information.

2.4.6 LBS service request processing

The information in this subsection is extracted from [14]

Considering the example of searching a Chinese restaurant the information chain from a service request to the answer will be described in the following and is illustrated in Figure 2-18.

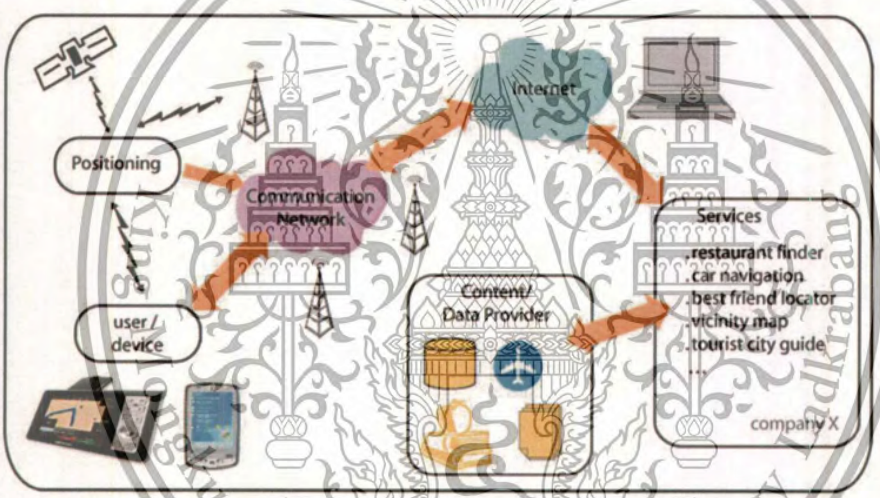


Figure 2.15 LBS components and information flow

2.4.7 Requirements of LBS architecture

The information in this subsection is extracted from [14]

Derived from the user actions, different requirements on the LBS system architecture emerge. Further different types of services are offered by companies to satisfy the needs. Whereas types of services will be described later we will start with the requirements on LBS.

Lists the following capabilities of LB-Services that usually exceed the general requirements on static GIS use:

- High Performance: Delivering answers in sub-second if querying information from Internet and databases.
- Scalable architecture: Support thousands of concurrent users and terabytes of data.
- Reliable: Capable of delivering up to 99.999 percent up time.
- Current: Support the delivery of real-time, dynamic information.
- Mobile: Availability from any device and from any location.
- Open: Support common standards and protocols
- Secure: Manage the underlying database locking and security services.
- Interoperable: Integrated with e-Business applications such as Customer Relationship Management, Billing, Personalization, and wireless positioning gateways.

These requirements lead to a complex LBS architecture involving a number of players. These Players include hardware and software vendors, content and online service providers, wireless network and infrastructure providers, wireless handset vendors and branded portal sites. Only common specifications and agreements among these players do ensure a user satisfying offering and deployment of services.

2.5 SQLite[15]

The information in this section is extracted from [15].

SQLite is an in-process library that implements a self-contained, serverless,

zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is currently found in more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - can be freely copied a database between 32-bit and 64-bit systems or between big endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen().

SQLite is a compact library. With all features enabled, the library size can be less than 350KB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) If optional features are omitted, the size of the SQLite library can be reduced below 200KB. SQLite can also be made to run in minimal stack space (4KB) and very little heap (100KB) making SQLite a popular database engine choice on memory constrained gadgets such as cellphones, PDAs, and MP3 players. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments.

SQLite is very carefully tested prior to every release and has a reputation for being

very reliable. Most of the SQLite source code is devoted purely to testing and verification. SQLite responds gracefully to memory allocation failures and disk I/O errors.

2.5.1 Features of SQLite

- Transactions are atomic, consistent, isolated, and durable (ACID) even after system crashes and power failures.
- Zero-configuration - no setup or administration needed.
- Implements most of SQL92. (Features not supported)
- A complete database is stored in a single cross-platform disk file.
- Supports terabyte-sized databases and gigabyte-sized strings and blobs.
- Small code footprint: less than 350KiB fully configured or less than 200KiB with optional features omitted.
- Faster than popular client/server database engines for most common operations.
- Simple, easy to use API.
- Written in ANSI-C. TCL bindings included. Bindings for dozens of other languages available separately.
- Well-commented source code with 100% branch test coverage.
- Available as a single ANSI-C source-code file that you can easily drop into another project.
- Self-contained: no external dependencies.
- Cross-platform: Unix (Linux and Mac OS X), OS/2, and Windows (Win32 and WinCE) are supported out of the box. Easy to port to other systems.

- Sources are in the public domain. Use for any purpose.
- Comes with a standalone command-line interface (CLI) client that can be used to administer SQLite databases.

2.5.2 Uses For SQLite

- **Application File Format**. Rather than using fopen() to write XML or some proprietary format into disk files used by your application, use an SQLite database instead to avoid having to write and troubleshoot a parser so that data will be more easily accessible and cross-platform, and the user's updates will be transactional.
- **Database For Gadgets**. SQLite is a popular choice for the database engine in cellphones, PDAs, MP3 players, set-top boxes, and other electronic gadgets. SQLite has a small code footprint, makes efficient use of memory, disk space, and disk bandwidth, is highly reliable, and requires no maintenance from a Database Administrator.
- **Website Database**. Because it requires no configuration and stores information in ordinary disk files, SQLite is a popular choice as the database to back small to medium-sized websites.
- **Stand-in For An Enterprise RDBMS**. SQLite is often used as a surrogate for an enterprise RDBMS for demonstration purposes or for testing. SQLite is fast and requires no setup, which takes a lot of the hassle out of testing and which makes demos perky and easy to launch.

Apple uses SQLite for many functions within Mac OS X, including Apple Mail,

Safari, and in Aperture. Apple uses SQLite in the iPhone and in the iPod touch and in iTunes software.

2.6 Objective-C [16]

The information in Objective-C section is extracted from [16]

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

Most object-oriented development environments consist of several parts:

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

2.6.1 The Runtime System

The Objective-C language defers as many decisions as it can from **compile time** and **link time** to **runtime**. Whenever possible, it dynamically performs operations such as creating objects and determining what method to invoke. Therefore, the language requires not just a compiler, but also a runtime system to execute the

compiled code. The runtime system acts as a kind of operating system for the Objective-C language; it is what makes the language work. Typically, however, developers do not need to interact with the runtime directly.

2.6.2 Syntax

Objective-C is a thin layer on top of C, and moreover is a *strict superset* of C; it is possible to compile any C program with an Objective-C compiler, and to freely include C code within an Objective-C class.

Objective-C derives its object syntax from Smalltalk. All of the syntax for non-object-oriented operations (including primitive variables, pre-processing, expressions, function declarations, and function calls) is identical to that of C, while the syntax for object-oriented features is an implementation of Smalltalk-style messaging.

- **Messages**

The Objective-C model of object-oriented programming is based on message passing to object instances. In Objective-C one does not simply *call a method*; one *sends a message*. This is unlike the Simula-style programming model used by C++. The difference between these two concepts is in how the code referenced by the method or message name is executed. In a Simula-style language, the method name is in most cases bound to a section of code in the target class by the compiler. In Smalltalk and Objective-C, the target of a message is resolved at runtime, with the receiving object itself interpreting the message. A method is identified by a *selector* or SEL — a NUL-terminated string representing its name — and resolved to a C method pointer implementing it: an IMP. A consequence of this is that the message-passing system has no type checking. The object to which the message is directed — the *receiver* — is not guaranteed to respond to a message, and if it does not, it simply raises an

exception.

Sending the message method to the object pointed to by the pointer obj would require the following code in C++:

```
obj->method(argument);
```

In Objective-C, this is written as follows:

```
[obj method:argument];
```

Both styles of programming have their strengths and weaknesses. Object-oriented programming in the Simula style allows multiple inheritances and faster execution by using compile-time binding whenever possible, but it does not support dynamic binding by default. It also forces all methods to have a corresponding implementation unless they are virtual, meaning the method is a placeholder for methods with the same name to be defined in objects derived from the base object. Smalltalk-style programming allows messages to go unimplemented, with the method resolved to its implementation at runtime. For example, a message may be sent to a collection of objects, to which only some will be expected to respond, without fear of producing runtime errors. Message passing also does not require that an object be defined at compile time. An implementation is still required for the method to be called in the derived object.

2.6.3 Interfaces and implementations

Objective-C requires that the interface and implementation of a class be in separately declared code blocks. By convention, developers place the interface in a header file and the implementation in a code file. The header files, normally suffixed .h, are similar to C header files while the implementation (method) files, normally

suffixed .m, can be very similar to C code files.

- **Interface**

The interface of a class is usually defined in a header file. A common convention is to name the header file after the name of the class. An interface declaration takes the form:

```
@interface classname : superclassname {
    // instance variables
}
+ classMethod1;
+ (return_type)classMethod2;
+ (return_type)classMethod3:(param1_type)param1_varName;

- (return_type)instanceMethod1:(param1_type)param1_varName :
(param2_type)param2_varName;
- (return_type)instanceMethod2WithParameter :
(param1_type)param1_varName
andOtherParameter:(param2_type)param2_varName;
@end
```

In the above, plus signs denote class methods, or methods that can be called on the class itself (not on an instance), and minus signs denote instance methods, which can only be called on a particular instance of the class. Class methods also have no access to instance variables.

The code above is roughly equivalent to the following C++ interface:

```
class classname : public superclassname {
 protected:
  // instance variables
 public:
  // Class (static) functions
  static void * classMethod1();
  static return_type classMethod2();
  static return_type classMethod3(param1_type param1_varName);
  // Instance (member) functions
  return_type instanceMethod1
(param1_type param1_varName, param2_type param2_varName);
  return_type instanceMethod2WithParameter
(param1_type param1_varName, param2_type param2_varName=default);
};
```

- **Implementation**

The interface only declares the class interface and not the methods themselves: the actual code is written in the implementation file. Implementation (method) files normally have the file extension .m, which originally signified "messages".

```
@implementation classname
+ (return_type)classMethod {
    // implementation
}
- (return_type)instanceMethod {
    // implementation
}
@end
```

Methods are written using their interface declarations. Comparing Objective-C and C:

```
- (int)method:(int)i {
    return [self square_root:i];
}
int function (int i) {
    return square_root(i);
}
```

```
 - (int)changeColorToRed:(float)red green:(float)green
blue:(float)blue;

[myColor changeColorToRed:5.0 green:2.0 blue:6.0];
```

The syntax allows pseudo-naming of arguments.

Internal representations of a method vary between different implementations of Objective-C. If myColor is of the class Color, instance method -changeColorToRed:green:blue: might be internally labeled _i_Color_changeColorToRed_green_blue. The i is to refer to an instance method, with the class and then method names appended and colons changed to underscores. As the order of parameters is part of the method name, it cannot be changed to suit coding style or expression as with true named parameters.

However, internal names of the function are rarely used directly. Generally, messages are converted to function calls defined in the Objective-C runtime library. It is not necessarily known at link time which method will be called because the class of the receiver (the object being sent the message) need not be known until runtime.

2.6.4 Protocols

Objective-C was extended at NeXT to introduce the concept of multiple inheritances of specification, but not implementation, through the introduction of protocols. This is a pattern achievable either as an abstract multiplies inherited base class in C++, or as an "interface" (as in Java and C#). Objective-C makes use of ad hoc protocols called *informal protocols* and compiler-enforced protocols called *formal protocols*.

An informal protocol is a list of methods that a class can opt to implement. It is specified in the documentation, since it has no presence in the language. Informal protocols often include optional methods, which, if implemented, can change the behavior of a class. For example, a text field class might have a delegate that implements an informal protocol with an optional method for performing auto-completion of user-typed text. The text field discovers whether the delegate implements that method (via reflection) and, if so, calls the delegate's method to support the auto-complete feature.

A formal protocol is similar to an interface in Java or C#. It is a list of methods that any class can declare itself to implement. Versions of Objective-C before 2.0 required that a class must implement all methods in a protocol it declares itself as adopting; the compiler will emit an error if the class does not implement every method from its

declared protocols. Objective-C 2.0 added support for marking certain methods in a protocol optional, and the compiler will not enforce implementation of optional methods.

The Objective-C concept of protocols is different from the Java or C# concept of interfaces, in that a class may implement a protocol without being declared to implement that protocol. The difference is not detectable from outside code.[*dubious – discuss*] Formal protocols cannot provide any implementations, they simply assure callers that classes that conform to the protocol will provide implementations. In the NeXT/Apple library, protocols are frequently used by the Distributed Objects system to represent the capabilities of an object executing on a remote system.

```
@protocol Locking
- (void)lock;
- (void)unlock;
@end
```

The syntax

Denotes that there is the abstract idea of locking. By stating that the protocol is implemented in the class definition:

Instances of SomeClass claim that they will provide an implementation for the two

```
@interface SomeClass : SomeSuperClass <Locking>
@end
```

instance methods using whatever means they choose. Another example use of abstract specification is describing the desired behaviors of plug-ins without constraining what the implementation hierarchy should be.

2.6.5 Language variants

- **Objective-C++**

Objective-C++ is a front-end to the GNU Compiler Collection, which can compile source files that use a combination of C++ and Objective-C syntax. Objective-C++ adds to C++ the extensions Objective-C adds to C. As nothing is done to unify the semantics behind the various language features, certain restrictions apply:

- A C++ class cannot derive from an Objective-C class and vice versa.
- C++ namespaces cannot be declared inside an Objective-C declaration.
- Objective-C declarations may appear only in global scope, not inside a C++ namespace
- Objective-C classes cannot have instance variables of C++ classes that do not have a default constructor or that have one or more virtual methods, but pointers to C++ objects can be used as instance variables without restriction (allocate them with new in the -init method).
- C++ "by value" semantics cannot be applied to Objective-C objects, which are only accessible through pointers.
- An Objective-C declaration cannot be within a C++ template declaration and vice versa. However, Objective-C types, (e.g., Classname *) can be used as C++ template parameters.
- Care must be taken since the destructor calling conventions of Objective-C and C++'s exception run-time models do not match. The new 64-bit runtime resolves this by introducing interoperability with C++ exceptions in this sense.

2.7 XML [17]

The information in this XML section is extracted from [17].

Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.

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

Many application programming interfaces (APIs) have been developed that software developers use to process XML data, and several schema systems exist to aid in the definition of XML-based languages.

As of 2009, hundreds of XML-based languages have been developed, including RSS, Atom, SOAP, and XHTML. XML-based formats have become the default for many office-productivity tools, including Microsoft Office (Office Open XML), OpenOffice.org (OpenDocument), and Apple's iWork.

2.7.1 Characters and escaping

XML documents consist entirely of characters from the Unicode repertoire. Except for a small number of specifically excluded control characters, any character defined by Unicode may appear within the content of an XML document. The selection of characters that may appear within markup is somewhat more limited but

still large.

XML includes facilities for identifying the encoding of the Unicode characters that make up the document, and for expressing characters that, for one reason or another, cannot be used directly.

The Unicode character set can be encoded into bytes for storage or transmission in a variety of different ways, called "encodings". Unicode itself defines encodings that cover the entire repertoire; well-known ones include UTF-8 and UTF-16. There are many other text encodings that pre-date Unicode, such as ASCII and ISO/IEC 8859; their character repertoires in almost every case are subsets of the Unicode character set.

XML allows the use of any of the Unicode-defined encodings, and any other encodings whose characters also appear in Unicode. XML also provides a mechanism whereby an XML processor can reliably, without any prior knowledge, determine which encoding is being used. Every XML parser will not necessarily recognize Encodings other than UTF-8 and UTF-16.

XML supports the direct use of almost any Unicode character in element names, attributes, comments, character data, and processing instructions (other than the ones that have special symbolic meaning in XML itself, such as the less-than sign, "<"). Therefore, the following is a well-formed XML document, even though it includes both Chinese and Cyrillic characters:

<?xml version="1.0" encoding="UTF-8" ?>

2.7.2 Use on the Internet

XML has come into common use for the interchange of data over the Internet. RFC 3023 gives rules for the construction of Internet Media Types for use when sending XML. It also defines the types "application/xml" and "text/xml", which say only that the data are in XML, and nothing about its semantics. The use of "text/xml" has been criticized as a potential source of encoding problems and is now in the process of being deprecated. RFC 3023 also recommends that XML-based languages be given media types beginning in "application/" and ending in "+xml"; for example "application/svg+xml" for SVG.

Further guidelines for the use of XML in a networked context may be found in RFC 3470, also known as IETF BCP 70; this document is very wide-ranging and covers many aspects of designing and deploying an XML-based language.

XML can also form a crucial part of a databaseless design.

2.7.3 Programming interfaces

The design goals of XML include, "It shall be easy to write programs which process XML documents." Despite this, the XML specification contains almost no information about how programmers might go about doing such processing. The XML Infoset specification provides a vocabulary to refer to the constructs within an XML document, but also does not provide any guidance on how to access this information. A variety of APIs for accessing XML have been developed and used, and some have been standardized.

Stream-oriented facilities require less memory and, for certain tasks, which are based on a linear traversal of an XML document, are faster and simpler than other alternatives. Tree-traversal and data-binding APIs typically require the use of much more memory, but are often found more convenient for use by programmers; some include declarative retrieval of document components via the use of XPath expressions.

XSLT is designed for declarative description of XML document transformations, and has been widely implemented both in server-side packages and Web browsers. XQuery overlaps XSLT in its functionality, but is designed more for searching of large XML databases.

2.8 KML[18]

Keyhole Markup Language (KML) is an XML notation for expressing geographic annotation and visualization within Internet-based, two-dimensional maps and three-dimensional Earth browsers. Originally named Keyhole Earth Viewer, KML was developed for use with Google Earth by Keyhole , Inc, which was acquired by Google in 2004. KML is an international standard of the Open Geospatial Consortium[18] . Google Earth was the first program able to view and graphically edit KML files. Other projects such as Marble [19] have also started to develop KML support.

2.8.1 Structure

The information in this sub section is extracted from [18].

The KML file specifies a set of features (place marks, images, polygons, 3D models, textual descriptions, etc.) for displaying in Google Earth, Maps and Mobile, or any other geospatial software implementing the KML encoding. Each place always has a longitude and a latitude. Other data can make the view more specifically, such as tilt, heading, altitude, which together define a "camera view". KML shares some of the same structural grammar as GML (Geography Markup Language) [20]. Some KML information cannot be viewed in Google Maps or Mobile. KML files are very often distributed in KMZ files, which are zipped files with a *.kmz* extension. These must be legacy (ZIP 2.0) compression compatible (i.e. stored or deflate method), otherwise the *.kmz* file might not uncompress in all geobrowsers. The contents of a KMZ file are a single root KML document (notionally "doc.kml") and optionally any overlays, images, icons, and COLLADA 3D models referenced in the KML including network-linked KML files. The root KML document is typically a file named "doc.kml" at the root directory level but the first *.kml* file entry in the KMZ file is the actual one

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Placemark>
  <name>New York City</name>
  <description>New York City</description>
  <Point>
    <coordinates>-74.006393,40.714172,0</coordinates>
  </Point>
</Placemark>
</Document>
</kml>
```

selected in Google Earth regardless of its name. By convention the root KML

document is at root level and referenced files are in subdirectories (e.g. images for overlay images). An example KML document is:

2.8.2 Geodetic reference systems in KML

The information in this sub section is extracted from [18].

For its reference system, KML uses 3D geographic coordinates: longitude, latitude and altitude, in that order, with negative values for west and south. The longitude, latitude components are as defined by the World Geodetic System of 1984 (WGS84). The vertical component (altitude) is measured from the WGS84 EGM96 Geoid vertical datum. If altitude is omitted from a coordinate string, e.g. (-122.917, 49.2623) then the default value of 0 (approximately sea level) is assumed for the altitude component, i.e. (-122.917, 49.2623, 0). A formal definition of the coordinate reference system (encoded as GML) used by KML is contained in the OGC KML 2.2 Specification. This definition references well-known EPSG CRS components.

Chapter 3

System Analysis and Design

3.1 System Analysis

In this chapter, we will give the analysis of our system and its design. Basically, there are three main functional requirements in our application. The first requirement is a function that allows a user to search for accommodations with three different searching types which are Search by Nearby, Search by Area and Searcy by Train station. The second requirement is to provide the search result in the list view and the map view including the information about each place. The third requirement is to provide a map which navigates the user from his current location to his chosen accommodation.

We will use Use case diagram and Activity, which are based on the UML structure as our tools to describe our system.

3.1.1 Use case diagram of our project

We will begin with the Use case to describe our system. Use case diagram is a simple diagram, which shows an overview of functional requirements of the system. That is, what the system can provide to its user. For our system, there are three functions: Search, Show on the map and Navigation.

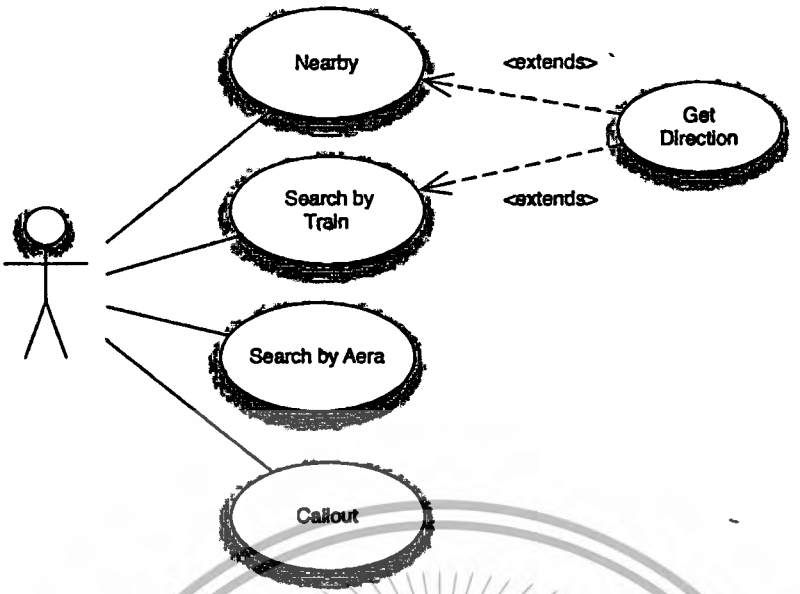


Figure 3.1 Application Use Case Diagram

3.1.2 Activity diagram of our project

Activity diagram is the diagram for showing the workflow of the system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagram describes the state of activities by showing the sequence of activities performed. We divide the participant into 3 parts: the user, the application and the Web service.

For our project, there are three Activity Diagrams. The Activity diagram of Search by Nearby, the Activity diagram of Search by Area and the last one is the Activity diagram of Search by Train station.

3.1.2.1 Activity diagram of Search by Nearby

Figure 3.2 shows the search by Nearby function. First when the user launches the application, the default page will be **Search by Nearby**, the system displays a list of accommodation types and the search radius to the user.

Then, the user will select the accommodation's type and adjust the search radius to the distance that the user wants. After the system receives the search request from the user, the system will connect and send the request to the web service, the web service will then connect to the webserver (www.home.co.th) and query the database according to the request. The webserver will provide the data that already filtered by KissXML class and send back to the system in XML format. When the system receives the data from the server the system will use a class name *KissXML* to extract data then display the accommodations' information to the user in the list view and map view. The user can further choose his interested house to see more detail. In this page, the system also provides the **Navigation button** to navigate the user to the selected accommodation. For the navigation function, system will connect and send the latitude and longitude of the user's current location to the Google Map API. Google will provide the dataset and send this data back to the system. When the system receives the data from Google, the system will use a class name *KMLParser* to extract data. In our system, we use only two-functions named *annotation* and *overlay* to display all possible route paths and the recommended path on the screen. In this page, the system also provides **Call button** for the user to make a phone call to the sale office from the application directly. For this calling function, the system will get a sale office's phone number from the database and use this number in method *showTel*.

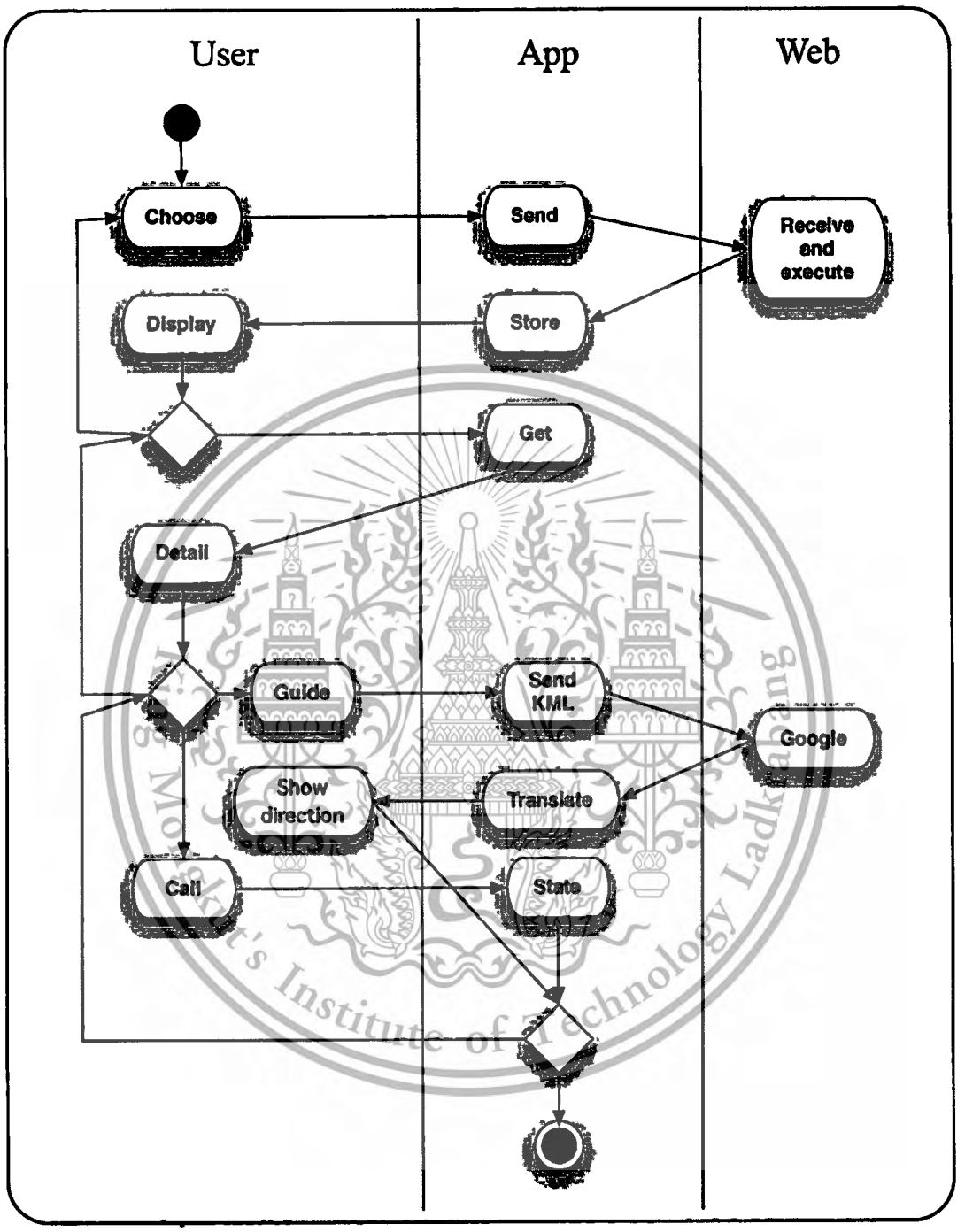


Figure 3.2 Activity Diagram of search by Nearby

3.1.2.2 Activity diagram of Search by Area

Figure 3.3 shows the search by Area function. First after the user launches the application and touches the **Area tab** on the tab bar at the bottom of the screen, the system displays a list of provinces, districts and accommodations' types to the user. Then, the user selects the province, the district, the type of the accommodation that he is interested in. Which in each request, after the system receives the search request from the user, the system will connect and send the request to the web service; the web service will connect to the webserver (*www.home.co.th*) and query the database according to the request. The webserver will provide the data that already filtered and send back to the system in XML format. When the system receives the data from the server the system will use a class name *KissXML* to filter data then displays the accomodation information to the user in the list view and map view. Then, the user chooses the interested house to see more detail. Similar to the previous search type, the system also provides **Call button** for the user to make a phone call to the sale office from the application directly.

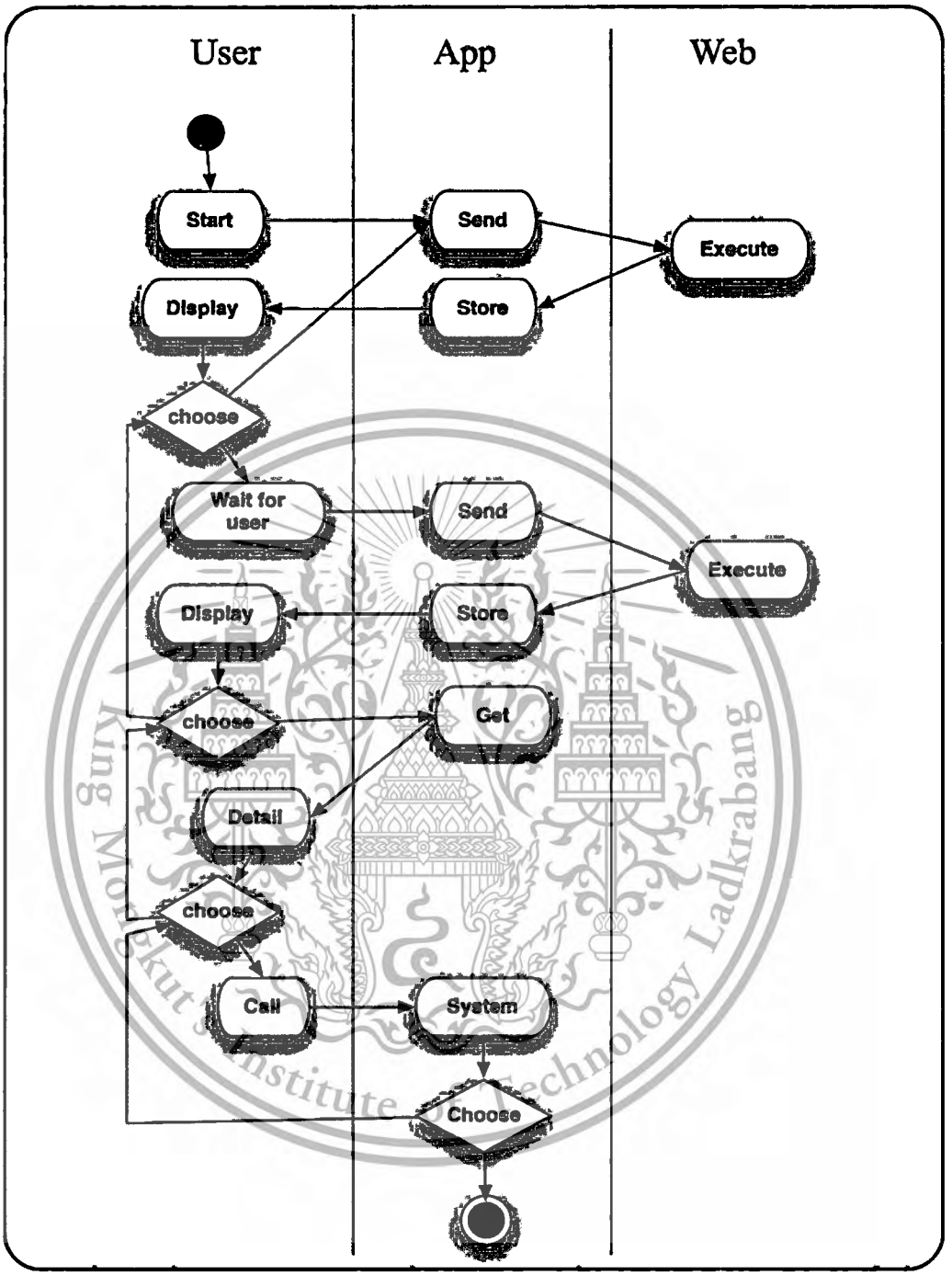


Figure 3.3 Search by Area activity diagram

3.1.2.3 Activity diagram of Search by Train

Figure 3.4 shows the search by Train function. First when the user launches the application, the system will query train information from the SQLite file stored in the application, then the user touches the **Train tab** on the tab bar at the bottom of the screen, the system displays a list of train services (MRT, BTS, or Airport link), stations and the types of houses to the user. The user then selects the train service, the station and the type of house that he is interested in. After the system receives the search request from the user, the system will connect and send search request to the web service, the web service will connect to the webserver (*www.home.co.th)* and query data from the database. The webserver will provide the data that query from its database and send back to the system in XML format. Note that in this type of search, the train information is stored on the phone using SQLite (since the list of train stations are rarely changed). The system obtains the train information by querying the SQLite database. For other information, the system will query from the web server through web service. When the system receives the data from the server the system will use a class name *KissXML* to filter data then display the houses' information to the user in list view and map view. The rest of the process is the same as described in Search by Nearby.

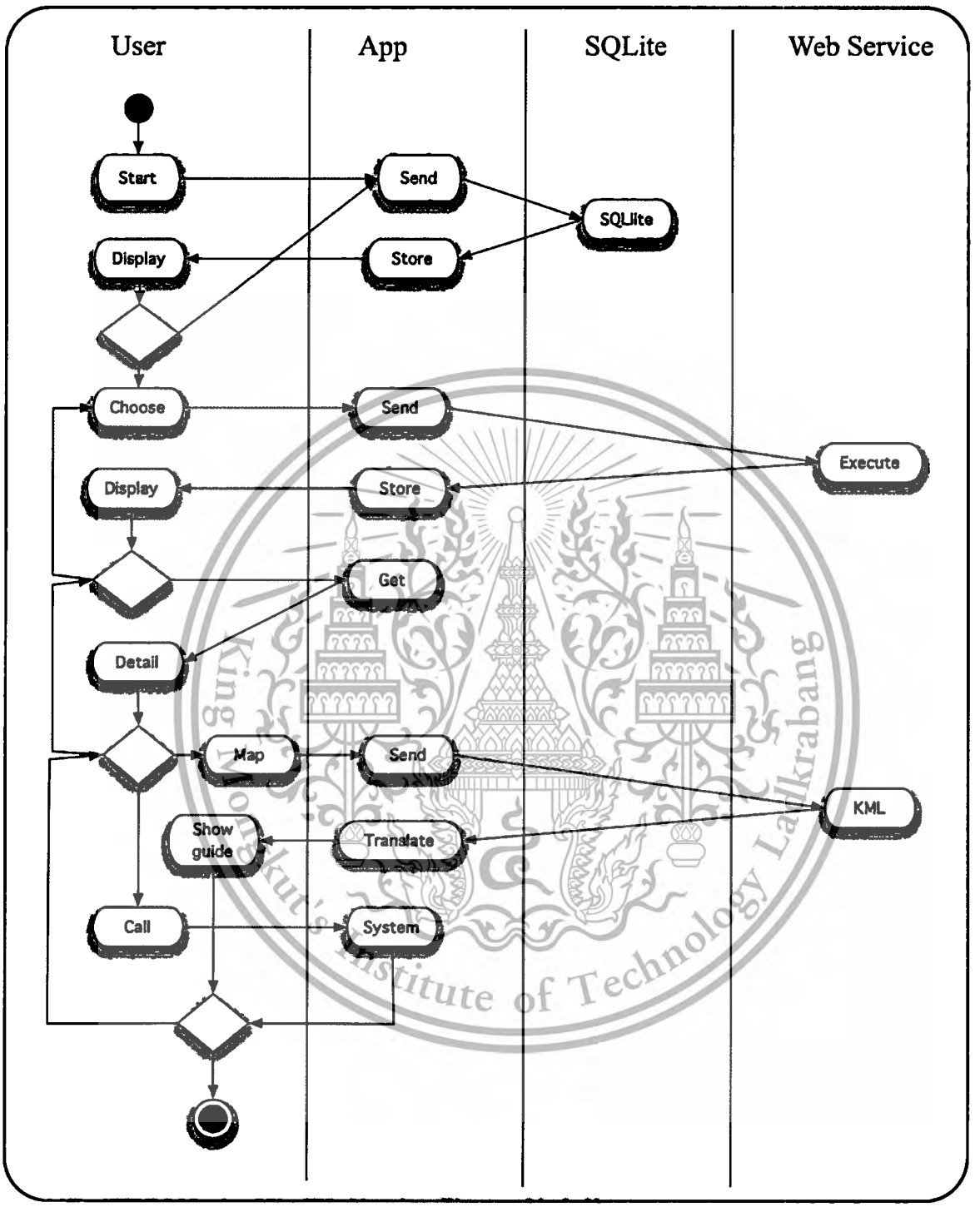


Figure 3.4 Search by Train activity diagram

3.2 System Design

3.2.1 Search method

This part is used to prepare data for a request sending to the web service, which includes the latitude and longitude of the starting point, the search radius and the accommodation type. Every search request from the system will be sent in the form of a soap message to the web service. Figure 3.5 at line 1 shows the URL of the server that we ask for its web service. Line 2 shows our soap message's components including the data that we want to ask from the web service such as latitude, longitude and radius etc. Line 3 is a code for our soap action. Line 4-12 shows the codes for preparing data before sending to the web service.

```
1    NSURL *url = [NSURL URLWithString:
     @"http://tamservice.home.do.th/MapNewHomeServices.svc"];
2    NSString    *soapMsg = [NSString stringWithFormat:@"<s:Envelope
     xmlns:s=\"http://schemas.xmlsoap.org/soap/envelope/\" ><s:Body><
     Get_NewHome_MAP_By_Lat_Lon_V3
     xmlns=\"http://tempuri.org/\"><userID>%@</userID><passWord>%@</passWord><choice>
     2</choice><latitude>%@</latitude><longtitude>%@</longtitude><radius>%@</radius><
     homeTypeCode>%@</homeTypeCode><linePerPage>%@</linePerPage><totalPage>0</
     totalPage><currentPage>1</currentPage><totalItem>0</totalItem></
     Get_NewHome_MAP_By_Lat_Lon_V3></s:Body></s:Envelope>",uCode,uPass,_lat,_lon,
     _radius,_hometype,kLinePerPage];
3    NSString    *soapAction =
     @"http://tempuri.org/IMapNewHomeServices/Get_NewHome_MAP_By_Lat_Lon_V3";
4    NSMutableURLRequest *req =    [NSMutableURLRequest requestWithURL:url];
5        NSString *msgLength =     [NSString stringWithFormat:@"%d", [soapMsg
         length]];
6        [req addValue:@"text/xml; charset=utf-8"
         forHTTPHeaderField:@"Content-Type"];
7        [req addValue:msgLength forHTTPHeaderField:@"Content-Length"];
8        [req setHTTPMethod:@"POST"];
9        [req setHTTPBody: [soapMsg dataUsingEncoding:NSUTF8StringEncoding]];
10       [req addValue:soapAction forHTTPHeaderField:@"SOAPAction"];
11       [appDelegate didStartNetworking];
12
13       conn = [[NSURLConnection alloc] initWithRequest:req delegate:self];
14       if (conn) {
15           webData = [[NSMutableData data] retain];
16       }
```

Figure 3.5 Search request

3.2.2 A method to filter data and move data to class name Place, Aumphur and Province.

When the system receives an XML data from the web service, the system first filters this data using the class named *KissXML* then stores in Model Class. In Figure 3.6, line 1-6 show a class named *DDXML* that we will use to filter data to store in the model after. Line 7 shows the method that we use to store data in a class named Place. Note that the class name *DDXML* is in the Library named *KissXML*.

```
1   DDXMLDocument *theDocument = [[DDXMLDocument alloc] initWithXMLString:source
    options:0 error:&error] ;
2      NSString *rootpath =@"//awst_GET_NEWHOMEDATA_MAP_V3";
3      static NSString *aBigName
       =@"http://www.home.co.th/images/img_v/search-list-thumb.jpg";
4
5      NSArray *resultNodes = [theDocument nodesForXPath:rootpath error:&error];
6
7         Place *aplace = [Place placeWithCoordinate:_location name:aName
          detail:aDesc hometype:ahometype imageName:aPicURL homeid:ahomeid
          largeImageName:aBigPic salesPrice:aPrice telNo:aTel
          locationAddress:aLocation companyName:aCompany custId:aCustID
          promotion:aPromotion distance:aDistance];
```

Figure 3.6 Filter Data to Class Model

3.2.3 A method to check the user current location

In **Search by Nearby**, the system must acquire the user current location. In Figure 3.7, **line 1** shows a **method that used to call a class name** ***CCLocationManager*** **(provided** by Xcode) **for checking the device location.** Line 2-3 shows codes for specifying the **accuracy of the device location.** Function ***startUpdatingLocation*** on line 5 is for checking the user's current location every time he launches the application or switches from another search type back to the nearBy search.

```
1   locationManager = [[CLLocationManager alloc] init];
2   locationManager.distanceFilter = 1.0;
3   locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
4   locationManager.delegate = self;
5   [locationManager startUpdatingLocation];
```

Figure 3.7 Check user current location

In **Search by Area**, the system will send the soap request message to the web service to ask for a list of provinces and a list of districts so that the user can choose and make a search order. Figure 3.8 from line 1 to 5 shows codes that are used to send the request soap message for the province code from the web service. Line 7-12 show the method used to request the list of districts and the latitude and longitude of each district to prepare for the required data .

```
NSURL *url = [NSURL URLWithString:
@"http://tamservice.home.co.th/MapNewHomeServices.svc"];

    NSString *soapMsg = [NSString stringWithFormat:@"<s:Envelope
    xmlns:s=\"http://schemas.xmlsoap.org/soap/envelope/\"><s:Body><
    Get_Province_NewHome_MAP
    xmlns=\"http://tempuri.org/\"><userID>%@</userID><passWord>%@</passWord><
    homeTypeCode/><linePerPage>%@</linePerPage><totalPage>0</totalPage><
    currentPage>0</currentPage><totalItem>0</totalItem></
    Get_Province_NewHome_MAP></s:Body></s:Envelope>",uCode,uPass,kLinePerPage];

    NSString *soapAction =
    @"http://tempuri.org/IMapNewHomeServices/Get_Province_NewHome_MAP";

    if ([dataType isEqualToString:@"2"]) {
        soapMsg =[NSString stringWithFormat:@"<s:Envelope
        xmlns:s=\"http://schemas.xmlsoap.org/soap/envelope/\"><s:Body><
        Get_Aumphur_NewHome_MAP
        xmlns=\"http://tempuri.org/\"><userID>%@</userID><passWord>%@</passWord>
        <provinceCode>%@</provinceCode><homeTypeCode/><linePerPage>%@</
        linePerPage><totalPage>0</totalPage><currentPage>0</currentPage><
        totalItem>0</totalItem></Get_Aumphur_NewHome_MAP></s:Body></s:Envelope>"
        ,uCode,uPass ,_province,kLinePerPage];

        soapAction =
        @"http://tempuri.org/IMapNewHomeServices/Get_Aumphur_NewHome_MAP";

    }
```

Figure 3.8 Sending a soap request to ask for provinces and districts

The method in Figure 3.9 shows codes used in setting the directory where the SQLite file are located.

```
+ (NSString *)dataFilePath {

    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    return [documentsDirectory stringByAppendingPathComponent:kFilename];
}
```

Figure 3.9 *dataFilePath* method

In **Search by Train**, the system will check the location of the SQLite file, open it, then **query the SQLite database's data (train information)** and store it in a class named *PointOfTrain* as shown in Figure 3.10. Then, the system will use a method named *getDatabase* to move data from *PointOfTrain* to an array which after that the system will use data in this array to execute the staring point of search by Train

In Figure 3.10, a method used to open the SQLite file and query data from it is shown. Line 12 calls *dataFilePath* method (see Figure 3.9). Line 16 shows the SQL statement for the train station information. Line 19-25 show codes used to convert received data from the SQLite database to strings which are stored in the *PointOfTrain* class.

```
7
8  + (NSMutableArray *)pointofTrain:trainLine {
9      [dataArray removeAllObjects];
10
11     sqlite3 *database;
12     if (sqlite3_open([[self dataFilePath] UTF8String], &database) != SQLITE_OK)
...    {
13         sqlite3_close(database);
14         NSAssert(0, @"Failed to open database");
15     }
16     NSString *query = [NSString stringWithFormat:@"SELECT
...    C_CODE,N_NO,C_STATION,C_LAT,C_LON FROM POINT_OF_TRAIN WHERE C_CODE='%@'
...    ORDER BY N_NO",trainLine];
17     sqlite3_stmt *statement;
18     if (sqlite3_prepare_v2(database, [query UTF8String], -1, &statement, nil) ==
...    SQLITE_OK) {
19         while (sqlite3_step(statement) == SQLITE_ROW) {
20
21             NSString *aCode = [NSString stringWithUTF8String:(char
...            *)sqlite3_column_text(statement, 0)];
22             NSString *aNo = [NSString stringWithUTF8String:(char
...            *)sqlite3_column_text(statement, 1)] ;
23             NSString *aStation = [NSString stringWithUTF8String:(char
...            *)sqlite3_column_text(statement, 2)];
24             NSString *aLat    = [NSString stringWithUTF8String:(char
...            *)sqlite3_column_text(statement, 3)];
25             NSString *aLon = [NSString stringWithUTF8String:(char
...            *)sqlite3_column_text(statement, 4)];
26
27             PointOfTrain *aplace = [PointOfTrain itemWithName:aCode atNo:aNo
...            atStation:aStation atLat:aLat atLon:aLon];
28
29             [dataArray addObject:aplace];
30         }
31         sqlite3_finalize(statement);     }
32
33     sqlite3_close(database);
34     return dataArray;
35 }
36
```

Figure 3.10 A method that download data from SQLite and store *PointOfTrain*

Figure 3.11 shows a method to display data obtaining from the SQLite database on the **Search page**. Line 43 shows a method that we will use to call data from class *TrainDB*. Codes on line 45 to 48 show how the data from *PointOfTrain* class such as **station's type, latitude and longitude** are put in the array.

```
37  -(void)getDataBase:(NSString *)dataType paraData:(NSString *)pcode{
38
39      if (self.trainStation ==nil) {
40
41      }else { [self.trainStation removeAllObjects];}
42
43      self.trainStation = [TrainDB pointofTrain:pcode];
44
45      if ([self.trainStation count]>0) {
46          PointOfTrain *train = [self.trainStation  objectAtIndex:0];
47          self.selStation = [NSMutableArray
...         arrayWithObjects:train.tCode,train.tStation,train.tLat,train.tLon, nil];
48      }
49      [self.tableView reloadData];
50
51  }
```

Figure 3.11 A method to move data from class *PointOfTrain* to array

In Figure 3.12, line 1-20 show a data model used to initialize the data format that we get from web service. Line 21-23 show the method to get result from every search type and put the data back to this model.

```
-(id)initWithCoordinate:(CLLocationCoordinate2D)c name:(NSString *)n
detail:(NSString *)d hometype:(NSString *)t imageName:(NSString *)p
homeid:(NSString *)h largeImageName:(NSString *)lp salesPrice:(NSString *)sp
telNo:(NSString *)tn locationAddress:(NSString *)la companyName:(NSString *)cn
custId:(NSString *)custI promotion:(NSString *)promoT distance:(NSString *)
disT{
    self = [super init];
    if (self) {
        self.coordinate = c;
        self.name = n;
        self.detail = d;
        self.hometype =t;
        self.imageName = p;
        self.homeid = h;
        self.largeImageName=lp;
        self.salesPrice=sp;
        self.telNo=tn;
        self.locationAddress=la;
        self.companyName=cn;
        self.custID = custI;
        self.promotion =promoT;
        self.distance = disT;
    }
    return self;
}
+(id)placeWithCoordinate:(CLLocationCoordinate2D)c name:(NSString *)n
detail:(NSString *)d hometype:(NSString *)t imageName:(NSString *)p
homeid:(NSString *)h largeImageName:(NSString *)lp salesPrice:(NSString *)sp
telNo:(NSString *)tn locationAddress:(NSString *)la companyName:(NSString *)cn
custId:(NSString *)custI promotion:(NSString *)promoT distance:(NSString *)
disT{
    return [[[Place alloc] initWithCoordinate:c name:n detail:d hometype:t
    imageName:p homeid:h largeImageName:lp salesPrice:sp telNo:tn
    locationAddress:la companyName:cn custId:custI promotion:promoT
    distance:disT] autorelease];
}
```

Figure 3.12 *Place* model class

Codes in Figure 3.13 are similar to codes in Figure 3.12 but used for storing received data for the search by train..

```
-(id)initWithName:(NSString *)_atCode atNo:(NSString *)_atNo atStation:(NSString
*)_atStation atLat:(NSString *)_atLat atLon:(NSString *)_atLon{
    self = [super init];
    if (self) {
        self.tCode = _atCode;
        self.tNo   = _atNo;
        self.tStation   = _atStation;
        self.tLat = _atLat;
        self.tLon = _atLon;

    }
    return self;
}

+(id)itemWithName:(NSString *)_atCode atNo:(NSString *)_atNo atStation:(NSString
*)_atStation atLat:(NSString *)_atLat atLon:(NSString *)_atLon{
    return [[[PointOfTrain alloc] initWithName:_atCode  atNo:_atNo
    atStation:_atStation atLat:_atLat atLon:_atLon] autorelease];
}
```

Figure 3.13 *PointOfTrain* class model

In Figure 3.14, line 1-4 show a method used to specify the number of sections. Line 6-16 are codes for finding the number of dataset (row) in each section.

```
1   - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
2   {
3       return 1;
4   }
5
6   - (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section
7   {
8       int count = [self.places count];
9
10      if (count == 0)
11      {
12          return kCustomRowCount;
13      }
14      return count;
15
16  }
```

Figure 3.14 Some codes in the table view management method

Figure 3.15 shows a method that used to manage the data in the list view page. Line 22 is a code for checking and counting the data that the system gets from the class name *Place*. A method on line 24-25 is for checking and counting the data which if no data I s not found, the no result message will be displayed. If the system finds the data (line 46), the system will get data from the class name *Place* (line 48). Line 49-50 shows code how the system gets the data from the entire row. Line 52-63 is codes for displaying the picture of each row. A class named *startIconDownload* is used to download each accommodation's picture. If the system does not finish downloading, a message "downloading…" is shown in the picture.

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

    static NSString *CellIdentifier = @"Cell";
    static NSString *PlaceholderCellIdentifier = @"PlaceholderCell";

    int nodeCount = [self.places count];

    if (nodeCount == 0 && indexPath.row == 0)
    {
        UITableViewCell *cell = [tableView
        dequeueReusableCellWithIdentifier:PlaceholderCellIdentifier];
        if (cell == nil)
        {
            cell = [[[UITableViewCell alloc]
            initWithStyle:UITableViewCellStyleSubtitle

                                          reuseIdentifier:
                                          PlaceholderCellIdentifier]
                                          autorelease];
            cell.detailTextLabel.textAlignment = UITextAlignmentCenter;
            cell.selectionStyle = UITableViewCellSelectionStyleNone;
        }

        cell.detailTextLabel.text = @"Loading ";

        return cell;
    }

    UITableViewCell *cell = [tableView
    dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil)
    {
        cell = [[[UITableViewCell alloc]
        initWithStyle:UITableViewCellStyleSubtitle
        reuseIdentifier:CellIdentifier] autorelease];
        cell.accessoryType = UITableViewCellAccessoryDetailDisclosureButton;
    }
    if (nodeCount > 0)
    {
        Place *appRecord = [self.places objectAtIndex:indexPath.row];
        cell.textLabel.text = appRecord.name;
        cell.detailTextLabel.text = appRecord.detail;

        if (!appRecord.appIcon)
        {
            if (self.tableView.dragging == NO && self.tableView.decelerating ==
            NO)
            {
                [self startIconDownload:appRecord forIndexPath:indexPath];
            }
            cell.imageView.image = [UIImage imageNamed:@"block_60.png"];
        }
        else
        {
            cell.imageView.image = appRecord.appIcon;
        }

    }
    return cell;
}
```

Figure 3.15 Codes showing how to store data in the tableview's row

Figure 3.16 is a method for managing the data in the map view page. Line 6 show a code, which the system get data from class named *Place.* Codes on line 13-33 are to control all annotations. The system will check data from the class named *Place* again on line 15. Line 18 show a code for setting an action of every pin (pop up window) while touching. Line 21-33 specifies the type of result such as house, condominium, or others. Line 38 – 52 show a method that sets all result to display on the screen. Line 57 – 58 show a method that sets the window type when the user clicks on each pin.

```
- (MKAnnotationView *)mapView:(MKMapView *)theMapView viewForAnnotation:(id
<MKAnnotation>)annotation
{
    if ([annotation isKindOfClass:[MKUserLocation class]])

        return nil;
    if ([annotation isKindOfClass:[Place class]])  // for City of San Francisco

    {
        static NSString *PlaceIdentifier = @"PlaceIdentifier";

        MKPinAnnotationView* pinView = (MKPinAnnotationView *)[mapView
        dequeueReusableAnnotationViewWithIdentifier:PlaceIdentifier];

        if (!pinView)
        {
            Place *place = (Place *)annotation;
            MKAnnotationView *annotationView = [[[MKAnnotationView alloc]
            initWithAnnotation:annotation reuseIdentifier:PlaceIdentifier]
            autorelease];

            annotationView.canShowCallout = YES;

            NSString *imgHome =@"";
            NSString *newHomeType = [place.hometype
            stringByTrimmingCharactersInSet: [NSCharacterSet
            whitespaceCharacterSet]];

            if ([newHomeType rangeOfString:@"Home"].location != NSNotFound) {
                imgHome =@"home_icon.png";
            }
            else {
                if ([newHomeType rangeOfString:@"Condo"].location != NSNotFound)
                {
                    imgHome =@"condo_icon.png";
                }
                else {
                    imgHome =@"others_icon.png";
                }
            }
            UIImage *flagImage = [UIImage imageNamed:imgHome];
            CGRect resizeRect;
            resizeRect.size = flagImage.size;

            CGSize maxSize = CGRectInset(self.view.bounds,[HomeMapVeiwController
            annotationPadding],[HomeMapVeiwController annotationPadding]).size;

            maxSize.height -=
            self.navigationController.navigationBar.frame.size.height +
            [HomeMapVeiwController calloutHeight];

            if (resizeRect.size.width > maxSize.width)
                resizeRect.size = CGSizeMake(maxSize.width,
                resizeRect.size.height / resizeRect.size.width * maxSize.width);

            if (resizeRect.size.height > maxSize.height)
                resizeRect.size = CGSizeMake(resizeRect.size.width /
                resizeRect.size.height * maxSize.height, maxSize.height);

            resizeRect.origin = (CGPoint){0.0f, 0.0f};
            UIGraphicsBeginImageContext(resizeRect.size);
            [flagImage drawInRect:resizeRect];
            UIImage *resizedImage = UIGraphicsGetImageFromCurrentImageContext();
            UIGraphicsEndImageContext();

            annotationView.image = resizedImage;
            annotationView.opaque = NO;

            UIButton *rightButton = [UIButton buttonWithType:
            UIButtonTypeInfoLight];
            annotationView.rightCalloutAccessoryView = rightButton;
            return annotationView;
        }
        else
        {
            pinView.annotation = annotation;
        }
```

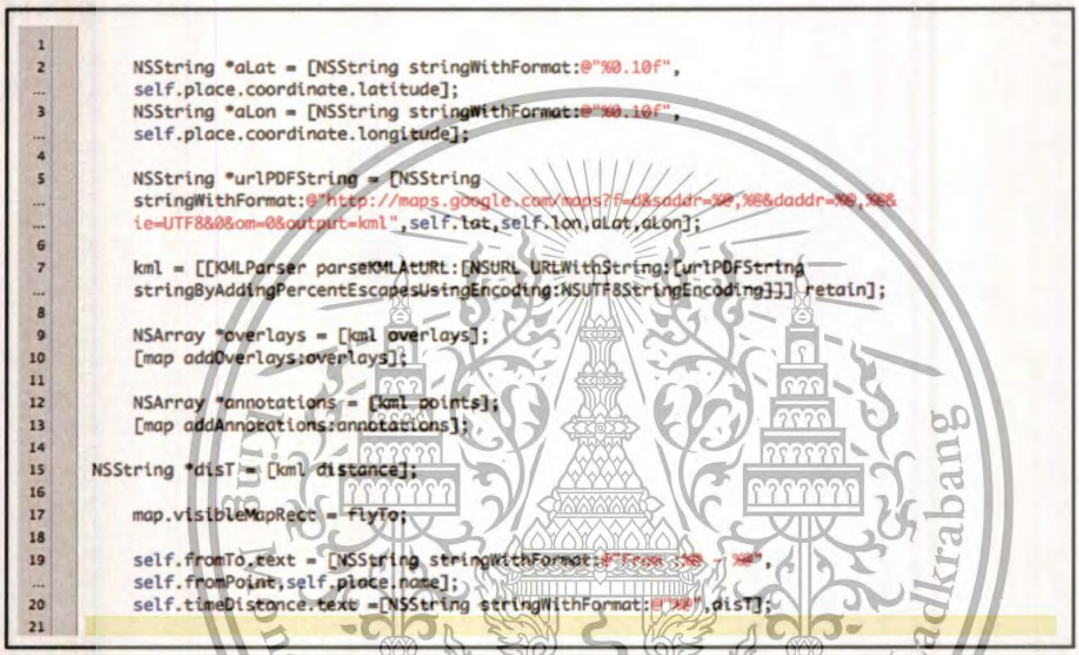
Figure 3.16 A method to displaying annotation on map view

Figure 3.17 shows a method that used to display the result at the detail view page. Line 12-20 show a method for getting information from the class named Place and displaying the information of selected accommodation. Line 22-29 is codes for showing the distance between the user current location or the train station location and the selected accommodation. Line 23 -25 are codes for search by area (unlike other search types, search by area will not display the distance). Line 27 -29 are codes for converting the unit from miles to kilometers distance. Codes from line 31 to 38 use the URL that we obtain from the class name named *Place* to download the accommodation's picture, which will be displayed on *UIImage* that already declared in the detail page.

```
12      self.title = self.place.name;
13      self.detailView.text = self.place.locationAddress ;
14      self.homeTypeView.text = self.place.hometype;
15      self.projectView.text = self.place.name;
16      self.telnoView.text = self.place.telNo;
17      self.priceView.text = self.place.salesPrice;
18      self.companyView.text = self.place.companyName;
19      self.coordinate = self.place.coordinate;
20      self.promotion.text = self.place.promotion;
21
22      NSString *resultString = [self.place.distance
        stringByTrimmingCharactersInSet: [NSCharacterSet
        whitespaceAndNewlineCharacterSet]];
23      if ([resultString isEqualToString:@""] || [resultString
        isEqualToString:@"0.00"]) {
24          self.distance.text =@"";
25          self.distance.hidden = YES;
26      } else {
27          double disT = [self.place.distance doubleValue];
28          disT = disT*1.61 ;
29          self.distance.text = [NSString stringWithFormat:@"Distance %0.2f km.
            \nFrom%@",disT,self.fromPoint];}
30
31      NSString * urlString = [self.place.largeImageName
        stringByAddingPercentEscapesUsingEncoding:NSASCIIStringEncoding];
32      NSURL * imageURL = [NSURL URLWithString:urlString];
33      NSData * imageData = [NSData dataWithContentsOfURL:imageURL];
34      UIImage *image = [UIImage imageWithData:imageData] ;
35      if (self.searchMode ==2) {
36          Mapbtn.hidden = YES;
37      }
38      self.imageView.image = image;
39
```

Figure 3.17 A method to display data on the detail view

In Figure 3.18, line 1-3 show how to convert latitude and longitude to **variable name** ***aLat*** **,** ***aLon*** before sending data to the Google Map API. Line 5 shows a variable name *urlPDFString* that the system uses for sending data to Google Map API. At line 7 variables *parseKMLAtURL* are declared and *KMLParser* class is used to send data to Google. Line 9 to 20 show a class that manages and display data on the screen.



```
1
2       NSString *aLat = [NSString stringWithFormat:@"%0.10f",
...     self.place.coordinate.latitude];
3       NSString *aLon = [NSString stringWithFormat:@"%0.10f",
...     self.place.coordinate.longitude];
4
5       NSString *urlPDFString = [NSString
...     stringWithFormat:@"http://maps.google.com/maps?f=d&saddr=%@,%@&daddr=%@,%@&
...     ie=UTF8&0&om=0&output=kml",self.lat,self.lon,aLat,aLon];
6
7       kml = [[KMLParser parseKMLAtURL:[NSURL URLWithString:[urlPDFString
...     stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]]] retain];
8
9       NSArray *overlays = [kml overlays];
10      [map addOverlays:overlays];
11
12      NSArray *annotations = [kml points];
13      [map addAnnotations:annotations];
14
15  NSString *disT = [kml distance];
16
17      map.visibleMapRect = flyTo;
18
19      self.fromTo.text = [NSString stringWithFormat:@"From %@ - %@",
...     self.fromPoint,self.place.name];
20      self.timeDistance.text =[NSString stringWithFormat:@"%@",disT];
21
```

Figure 3.18 request data from Google map API method

3.3 Table on SQLite File

The database that we use to store the information of the train stations is stored in the file named *housefinder.sqlite*. There is one table in our database. That is **"PointOfTrain"**. The fields in the table are the station ID, type of train (BTS or MRT or APL), the name of each station, the order of the station, the latitude of each station and the longitude of each station. The data inside this table is used in the Search by train for the train information. When the user already selects the station, the latitude and longitude of that station will be sent to the web service so the web service can use these locations to search for the accommodations. See Table 3.1 for the table detail.

Table 3.1 PointOfTrain table

Name	Type	Description	Key
ID	INTEGER	Train ID	Primary key
C_CODE	TEXT	Type of Station	
C_STATION	TEXT	**Station's name**	
N_NO	INTEGER	Order of each station	
C_LON	TEXT	Latitude of the station	
C_LAT	TEXT	Longitude of the station	

3.4 The data that receive from web service

There are three types of messages that our application received from the web service. The first type is the message that the web service sends back for the search query. It contains information about the places (Place Class). All searching types will return this place class message after the user has submitted his query. Table 3.2 below shows the data that the system receives from the web service. The second type (Province Class) and the third type (Aumphur Class) of the message are used in the Search by Area option for the list of provinces and the list of that province's districts (Aumphur). After the system receives the data from the web service, the system will convert each message tag to a string data and stored this data in the array of each class except P_MARK_LAT and P_MARK_LON which the system will combine P_MARK_LAT and P_MARK_LON together then add to a function named *CLLocationCoordinate2D* for using with map view on Xcode.

1. Place Class: this class is used to store the search result of the accommodation of all search type including ID, Name, Picture, URL, Price, etc. for using in the list View, Map view and Detail view as shown in Table 3.2

Table 3.2 Place table

Name	Type	Description
A_ID	INT	House ID
C_PROJECT_NAME	NAVARCHAR(100)	House 's name
C_PROJECT_NAME_E	NAVARCHAR(100)	House's detail
C_PRJ_TYPE_ID	NAVARCHAR(20)	House's type
C_PRJ_PIC	NAVARCHAR(100)	Image name
C_PROMOTION	NAVARCHAR(100)	Promotion
C_PRJ_PIC_PLAN	NAVARCHAR(100)	Large image name
C_PRICE	NAVARCHAR(250)	Sale price
C_PRJ_TELEPHONE	NAVARCHAR(100)	Telephone number
C_PRJ_ADDRESS	NAVARCHAR(250)	Location address
C_CUSTNAME_T	NAVARCHAR(100)	Company's name
N_CUST_ID	INT	Customer ID
P_MARK_LON	NAVARCHAR(50)	Longitude
P_MARK_LAT	NAVARCHAR(50)	Latitude

2. Province Class: the system will get the list of the province codes (C_PROVINCE_CODE) and names (C_PROVINCE_NAME) as shown in Table 3.3. This information is used in the province drop down list.

Table 3.3 Province table

Name	Type
C_PROVINCE_CODE	NAVARCHAR(4)
C_PROVINCE_NAME	NAVARCHAR(50)

3. Aumphur Class: the system will get the list of the district codes (C_AMPOR_CODE) and names (C_AMPOR_NAME). This information is used in the Aumphur drop down list to display list of districts in the user selected province. See Table 3.4 for more fields in Aumphur table.

Table 3.4 Aumphur table

Name	Type
C_AMPOR_CODE	NAVARCHAR(4)
C_AMPOR_NAME	NAVARCHAR(50)
C_PROVINCE_CODE	NAVARCHAR(4)
C_PROVINCE_NAME	NAVARCHAR(50)
C_MARK_LAT	NAVARCHAR(50)
C_MARK_LON	NAVARCHAR(50)

Chapter 4

Implementation

4.1 iOS Application

The "Home Finder" application is developed for searching currently available accommodations for sale that located either nearby the user's position or the position which the user chooses, there are 3 different search types including **Search Nearby**, **Search by Area**, and **Search by Train** (BTS/MRT/Airport Link) station. The application displays the search result on the list view and the map view. More detail of each house and some useful information are also provided so that it can help the user making his decision easier.

4.2 User Interface

4.2.1 Nearby search view

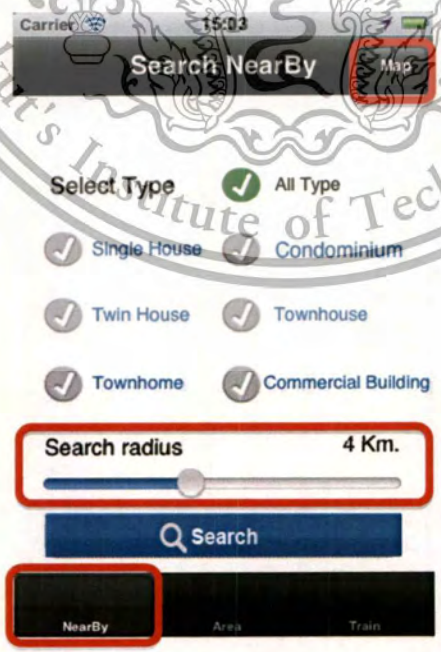


Figure 4.1 Search Nearby Page

Figure 4.1 shows the search nearby page which allows a user to choose the type of accommodations that he wants (see Figure 4.2 line 1-26) and he can specify the search radius from his current position (see Figure 4.2 line 28-31). The search result will be displayed in two types, the list view (Figure 4.7) and the map view (Figure 4.11).

```
1   - (IBAction)checkAction1:(id)sender
2   {
3       if (self.button1Selected == 0)
4       {
5           [checkButton1 setSelected:YES];
6           self.button1Selected = 1;
7       }
8       else
9       {
10          [checkButton1 setSelected:NO];
11          self.button1Selected = 0;
12          [checkButton2 setSelected:NO];
13          self.button2Selected = 0;
14          [checkButton3 setSelected:NO];
15          self.button3Selected = 0;
16          [checkButton4 setSelected:NO];
17          self.button4Selected = 0;
18          [checkButton5 setSelected:NO];
19          self.button5Selected = 0;
20          [checkButton6 setSelected:NO];
21          self.button6Selected = 0;
22          [checkButton7 setSelected:NO];
23          self.button7Selected = 0;
24
25      }
26  }
27
28  - (IBAction)ValueSliderChanged:(UISlider *)sender {
29
30      radiusValueLabel.text = [NSString stringWithFormat:@"%.0f km.",
    ...
31      sender.value];
32  }
```

Figure 4.2 IBAction for checking which house type that user chooses

4.2.2 Area search view

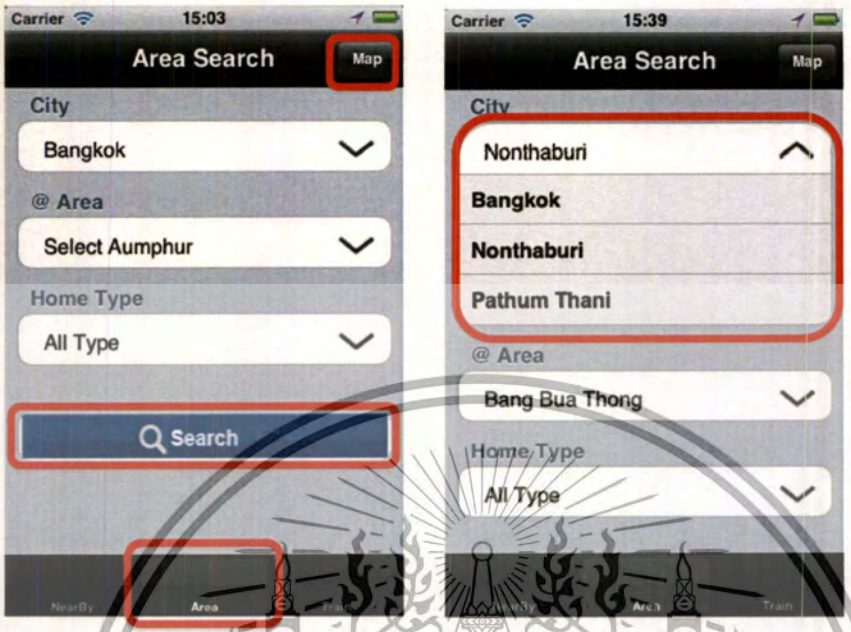


Figure 4.3 Search by area page

Figure 4.3 shows the Search by Area page which the user first needs to choose the province, the Aumphur and the accommodation type. The search result will display in two types, map view and list view as shown in Figure 4.7 and 4.11.

In Figure 4.4 line 1-4 display codes that specify a number of sections for creating the drop-down menu in Search by Area and Search by Train pages. Codes in line 6-33 specify the numbers of rows in each section and 1 is added (+1) for the selected item of each menu.

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 4;
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    switch (section) {
        case 0: {

                if (dropDown1Open) { return [self.provinceData count]+1; }
                else  {  return 1;}
                break;
        }
        case 1:
        {

                if (dropDown2Open) { return [self.aumphurData count]+1;}
                else    { return 1;  }

            break;
        }

        case 2:{
            if (dropDown3Open) { return [self.homeType count]+1 ;  }
            else
            {    return 1;     }
            break;}
        default:
            return 1;
            break;
    }
}
```

Figure 4.4 A method for setting numbers of section in table view

4.2.3 Train search view

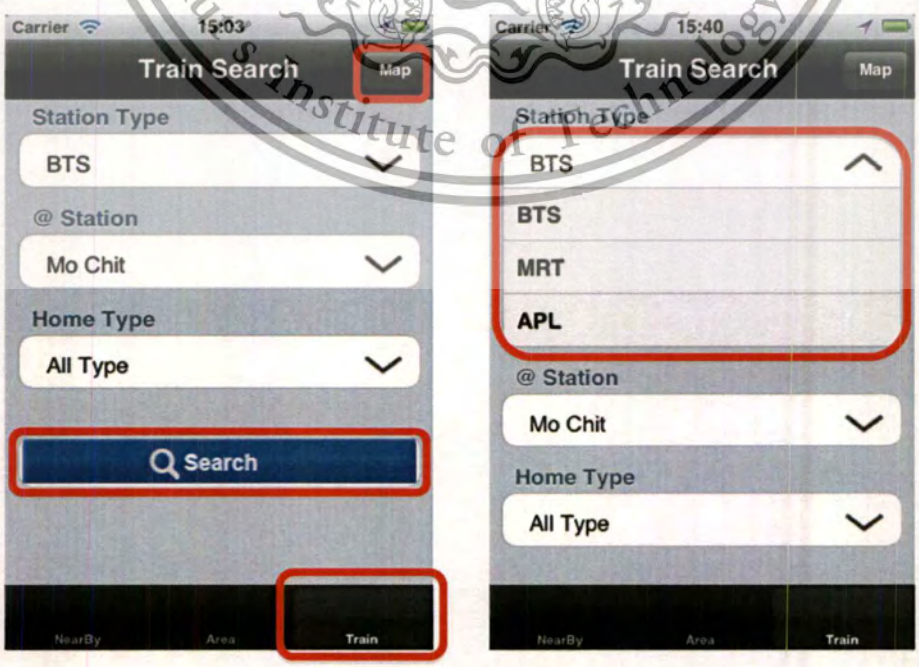


Figure 4.5 Search by train station page

Figure 4.5 shows Search by Train station, in this page the application will ask the user to select the type of the train station, the name of the station, the accommodation type. The default radius of the train type search is 2 kilometers. The search result will be displayed in two types, the map view and the list view as shown in Figure 4.7 and 4.11.

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath
{
    int icount =0;
    switch ([indexPath section]) {
        case 0: {

            switch ([indexPath row]) {
                case 0:
                {
                    DropDownCell *cell = (DropDownCell*) [self.tableView
                    cellForRowAtIndexPath:indexPath];

                        icount = [trainline count];

                    NSMutableArray *temp= [[NSMutableArray alloc]
                    initWithObjects: nil];
                    for (int i=0;i<icount;i++){
                        NSIndexPath *path = [NSIndexPath
                        indexPathForRow:[indexPath row]+i+1 inSection:indexPath
                        section]];
                        [temp addObject:path];
                    }

                    NSArray *indexPathArray = [NSArray  arrayWithArray:temp];
                    [temp release];

                    if (dropDownIOpen == YES) //   ([cell isOpen])
                    {
                        [cell setClosed];
                        dropDownIOpen =  NO;//[cell isOpen];
                        [self.tableView deleteRowsAtIndexPaths:indexPathArray
                        withRowAnimation:UITableViewRowAnimationTop];
                    }
                    else
```

Figure 4.6 A method to store and display the data in each section

```
                NSMutableArray *temp= [[NSMutableArray alloc]
                initWithObjects: nil];
                for (int i=0;i<icount;i++){
                    NSIndexPath *path = [NSIndexPath
                    indexPathForRow:[indexPath row]+i+1 inSection:[indexPath
                    section]];
                    [temp addObject:path];
                }

                NSArray *indexPathArray = [NSArray  arrayWithArray:temp];
                [temp release];
                //
                if (dropDown1Open == YES) //   ([cell isOpen])
                {
                    [cell setClosed];
                    dropDown1Open =  NO;//[cell isOpen];
                    [self.tableView deleteRowsAtIndexPaths:indexPathArray
                    withRowAnimation:UITableViewRowAnimationTop];
                }
                else

                {
                    [cell setOpen];
                    dropDown1Open = YES;//[cell isOpen];

                    [self.tableView insertRowsAtIndexPaths:indexPathArray
                    withRowAnimation:UITableViewRowAnimationTop];
                }
                break;
            }
            default:
            {
                dropDown1 = [[[self.tableView
                cellForRowAtIndexPath:indexPath] textLabel] text];


                icount = [trainline count];
                if (icount>0) {
                    self.selTrainline = [self.trainline
                    objectAtIndex:indexPath.row-1];}

                NSIndexPath *path = [NSIndexPath indexPathForRow:0
                inSection:[indexPath section]];
                DropDownCell *cell = (DropDownCell*) [self.tableView
                cellForRowAtIndexPath:path];

                [[cell textLabel] setText:dropDown1];
                //- Close
                NSMutableArray *temp= [[NSMutableArray alloc]
                initWithObjects: nil];
                for (int i=0;i<icount;i++){
                    NSIndexPath *path = [NSIndexPath indexPathForRow:i+1
                    inSection:[indexPath section]];

                    [temp addObject:path];
                }
                NSArray *indexPathArray = [NSArray  arrayWithArray:temp];
                [temp release];
                [cell setClosed];
                dropDown1Open = NO;//[cell isOpen];

                [self.tableView deleteRowsAtIndexPaths:indexPathArray
                withRowAnimation:UITableViewRowAnimationTop];

                    //- Load Station of Train

                    [self getDataBase:@"POINT_OF_TRAIN"
                    paraData:self.selTrainline];

                break;
            }
        }

        break;
    }
```

Figure 4.7 A method to store and display the data in each section Cont.

Figure 4.6 is a method that used to display the dropdown list when the user clicks at the station type on the search by train view. Line 8 to line 37 show an action when the user touches on the station type menu which the application will provide the dropdown list. Line 23-28 are codes that used to pull up the dropdown menu. Line 29 to line 35 are codes that used to pull down the dropdown menu. Line 38 to line 72 show an action after the user touches on each row in the dropdown list. Line 64 to line 68 show an action after the user selects the dropdown menu.

4.2.4 Search result on the list view

Figure 4.8 Search result on list view

Figure 4.7 shows the search result from the search page, which is displayed in the list view and allows the user to click on any list item to get more information about each accommodation. In this page, the user can go back to the search page to search again or touch on the **Map tap** to see the search result in the map view (see Figure 4.11). **Line 3** is a code for creating an **array named** ***homelist*** which **stores data from** the ***Place*** class (see Section 3.4). Line 4 calls a **class named** *HomeMapViewController* with parameters such as an array *homelist* , *seachMode*, *Lat* and *Lon*. Line 5 sets the ***navigationController*** to the **Map** view page with all values from line 4.

```
1   - (void) MapViewed: (id)sender
2   {
3       NSArray *homelist = self.places;
4       HomeMapVeiwController *homeListViewController = [[HomeMapVeiwController
        alloc] initWithPlace:homelist fromMode:1 searchMode:self.searchMode
        Lat:self.lat Lon:self.lon fromPoint:self.fromPoint];
5       [self.navigationController pushViewController:homeListViewController
        animated:YES];
6       [homeListViewController release];
7   }
```

Figure 4.9 A method to go to the map guide view

Figure 4.9 from line 8 to 17 shows a method used to add an action when the user touches on each row (which will direct to the Detail page (see Figure 4.15)). Line 19-26 show codes when the user touches on which will also go to the Detail page.

```
8   - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
    *)indexPath
9   {
10      Place *place = [self.places objectAtIndex:indexPath.row];
11
12      HomeDetailViewController *homeViewController = [[HomeDetailViewController
        alloc] initWithPlace:place fromMode:1 searchMode:self.searchMode
        Lat:self.lat Lon:self.lon fromPoint:self.fromPoint];
13
14      [self.navigationController pushViewController:homeViewController
        animated:YES];
15      [homeViewController release];
16
17  }
18
19  - (void)tableView:(UITableView *)tableView
    accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath{
20
21      Place *place = [self.places objectAtIndex:indexPath.row];
22
23      HomeDetailViewController *homeViewController = [[HomeDetailViewController
        alloc] initWithPlace:place  fromMode:1 searchMode:self.searchMode
        Lat:self.lat Lon:self.lon fromPoint:self.fromPoint];
24      [self.navigationController pushViewController:homeViewController
        animated:YES];
25      [homeViewController release];
26  }
```

Figure 4.10 Actions when accommodation on the list view is selected

Figure 4.10 shows codes that the list view page uses to receive data from the search page. Line 27 shows *initWithPlace* method which takes the search result from parameters such as *_homelist*, *_Lat* and *_Lon* and stores in the program variables (see line 29-35). Note that we store the current location of the phone in *_Lat* and *_Lon*.

```
27  -(id)initWithPlace:(NSArray *)__homeList searchMode:(int)__searchMode
    Lat:(id)__Lat Lon:(id)__Lon fromPoint:(NSString *)__point{
28      self = [super initWithNibName:@"HomeListViewController" bundle:nil];
29      if (self) {
30          self.places= [NSArray arrayWithArray: __homeList] ;
31          self.lat = __Lat;
32          self.lon = __Lon;
33          self.searchMode = __searchMode;
34          self.fromPoint =__point;
35      }
36      return self;
37  }
```

Figure 4.11 A method to get data from Class model

4.2.5 Search result on the map view

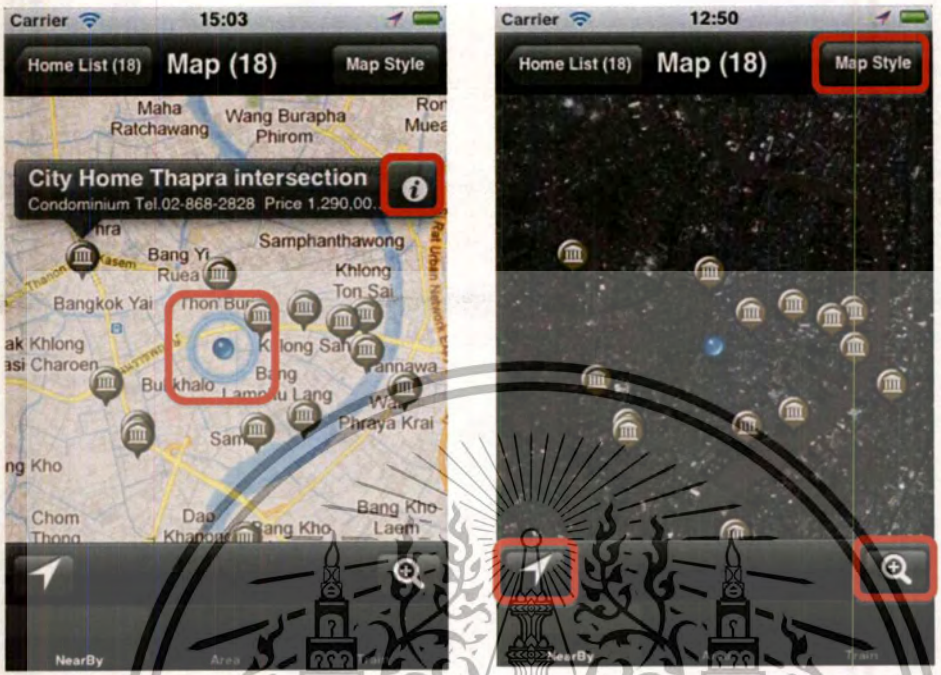


Figure 4.12 Search result on map view in standard and satellite

Figure 4.11 shows the result from the search page, which is displayed on the map view. This page will show the user's position and all positions of accommodations within in the specified radius. Different symbols are used to represent each type of accommodation. The user can zoom in around his current position by pressing  and can set the center of the map to his current position by pressing  . He can also switch to the satellite map mode by touching **Map Style** button.

```
1   -(id)initWithPlace:(NSArray *)__homeList fromMode:(int)__mode
    searchMode:(int)__searchMode Lat:(id)__Lat Lon:(id)__Lon fromPoint:(NSString
    *)__point{
2       self = [super initWithNibName:@"HomeMapVeiwController" bundle:nil];
3       if (self) {
4           self.places=  __homeList ;
5           self.lat = __Lat;
6           self.lon = __Lon;
7           self.searchMode = __searchMode;
8           self.fromPoint =__point;
9       }
10      if (self.places.count==0)
11      {
12          UIAlertView * alertView = [[UIAlertView alloc] initWithTitle:@"Home
            Result:" message:@"No Data" delegate:nil cancelButtonTitle:@"Dismiss"
            otherButtonTitles:nil];
13          [alertView show];
14          [alertView release];
15      }
16      return self;
17  }
```

Figure 4.13 A method for getting data from class model to display pin on the screen

Figure 4.12 shows codes which the map view page uses to receive from the search page. If no result is found, there is a pop-up window to alert the user (see line 10-15).

```
19  -(void)mapView:(MKMapView *)mapView annotationView:(MKAnnotationView *)view
    calloutAccessoryControlTapped:(UIControl *)control{
20
21      Place *place =(Place *)view.annotation;
22      if ([place  isKindOfClass:[Place class]]){
23
24          HomeDetailViewController *homeViewController =
            [[HomeDetailViewController  alloc] initWithPlace:place fromMode:2
            searchMode:self.searchMode Lat:self.lat Lon:self.lon
            fromPoint:self.fromPoint];
25          [self.navigationController pushViewController:homeViewController
            animated:YES];
26          [homeViewController release];
27
28      }
29  }
```

Figure 4.14 Actions when user touch the button on map view

Figure 4.13 shows codes which add an action to the symbol. When the user clicks on each symbol, it will go to the **Detail page** as in Figure 4.9.

In Figure 4.14, *ChangeMapType* function on the map view page from line 31-37 is for changing the map mode between the standard and the satellite modes. Line 39-44 are codes for zooming into the user current location that shows on the map. Line 45-47 perform the same function as line 39-44 without zooming in.

```
- (void) changeMapType: (id)sender
{
    if (mapView.mapType == MKMapTypeStandard)
        mapView.mapType = MKMapTypeSatellite;
    else
        mapView.mapType = MKMapTypeStandard;
}

- (void)zoomIn: (id)sender
{
    MKUserLocation *userLocation = mapView.userLocation;
    MKCoordinateRegion region = MKCoordinateRegionMakeWithDistance
    (userLocation.location.coordinate, 1000, 1000);
    [mapView setRegion:region animated:NO];
}
- (IBAction)showCurrentLocation:(id)sender {
    [mapView setRegion:MKCoordinateRegionMake([mapView.userLocation coordinate],
    MKCoordinateSpanMake(0.05, 0.05)) animated:YES];
}
```

Figure 4.15 Functions on map view (change map type, zoom in, go to current location)

4.2.6 Detail view

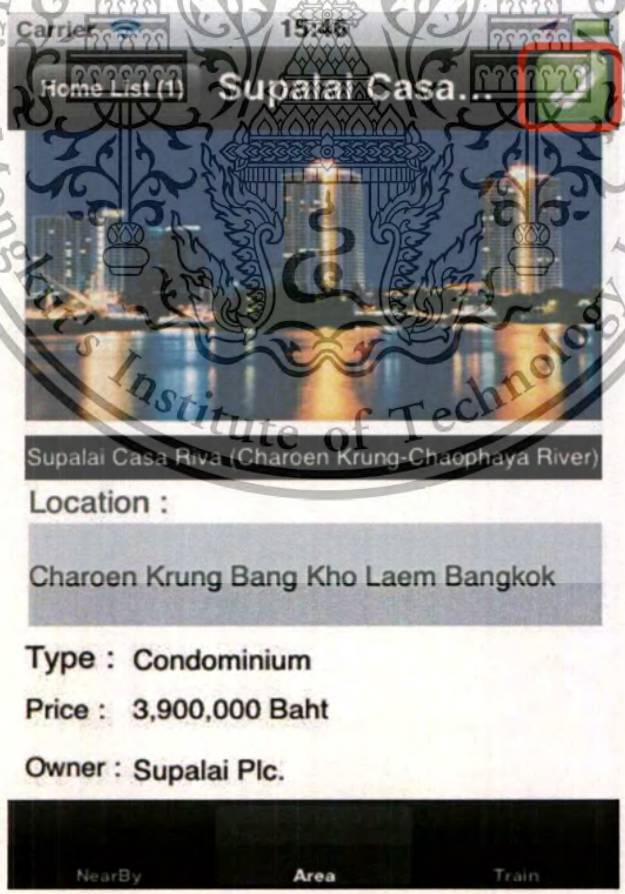


Figure 4.16 An example of the Detail page

Figure 4.15 show information about each accommodation that the user chooses from the list view. This information includes picture, location, type, and price. The user can go back to the list view page for other accommodations or can call to ask for more information directly from the sale office.

Figure 4.16 shows codes for sending data from he list view or the map view page to the **Detail page**.

```
-(id)initWithPlace:(Place *)__place fromMode:(int)__mode
searchMode:(int)__searchMode Lat:(NSString *)__Lat Lon:(NSString *)__Lon
fromPoint:(NSString *)__point{
    self = [super initWithNibName:@"HomeDetailViewController" bundle:nil];
    if (self) {
        self.place = __place ;
        self.lat = __Lat;
        self.lon = __Lon;
        self.searchMode = __searchMode;
        self.fromPoint = __point;
    }
    return self;
}
```

Figure 4.17 A method for getting data from the class model

Figure 4.17 show codes for going to the map guide page by using navigation controller. The starting point of the path is either the user current location (Search by Nearby) or the train station (Search by Train).

```
-(IBAction)showMap:(id)sender{

    MapGuideViewController *homeViewController = [[MapGuideViewController alloc]
    initWithPlace:self.place fromMode:1 searchMode:self.searchMode Lat:self.lat
    Lon:self.lon fromPoint:self.fromPoint ];
    [self.navigationController pushViewController:homeViewController
    animated:YES];
    [homeViewController release];

}
```

Figure 4.18 A method to go to map guide view

Figure 4.18 shows a method used for creating a button on the navigation bar in our application. There are 2 buttons: the callout button  and the **Mapguide** button. Both buttons are on the same position. The callout button (*searchMode==2*) will appear only in the Search by Area screen (see line 1-11). The **Mapguide** button will appear in the Search by Nearby and Search by Train screens (see line 12-15). Note that in Search by Area page our application does not provide the **Mapguide** button.

```
1  if (self.searchMode ==2) {
2          UIButton *localFlipIndicator=[[UIButton alloc]
           initWithFrame:CGRectMake(0,0,40,40)];
3          self.flipIndicatorButton=localFlipIndicator;
4          [localFlipIndicator release];
5          // front view is always visible at first
6          [self.flipIndicatorButton setBackgroundImage:[UIImage
           imageNamed:@"Phone01.png"] forState:UIControlStateNormal];
7          UIBarButtonItem *flipButtonBarItem;
8          flipButtonBarItem=[[UIBarButtonItem alloc]
           initWithCustomView:flipIndicatorButton];
9          [self.navigationItem setRightBarButtonItem:flipButtonBarItem
           animated:YES];
10         [flipButtonBarItem release];
11         [flipIndicatorButton addTarget:self action:@selector(showTel:)
           forControlEvents:(UIControlEventTouchDown)];
12     }else  {
13         UIBarButtonItem *addBtn = [[UIBarButtonItem alloc] initWithTitle:@"Map"
           style:UIBarButtonItemStyleBordered  target:self
           action:@selector(showMap:)];
14         self.navigationItem.rightBarButtonItem = addBtn;
15         [addBtn release];
16     }
17
```

Figure 4.19 A method for creating a button on navigation bar

4.2.7 Detail page with map guide

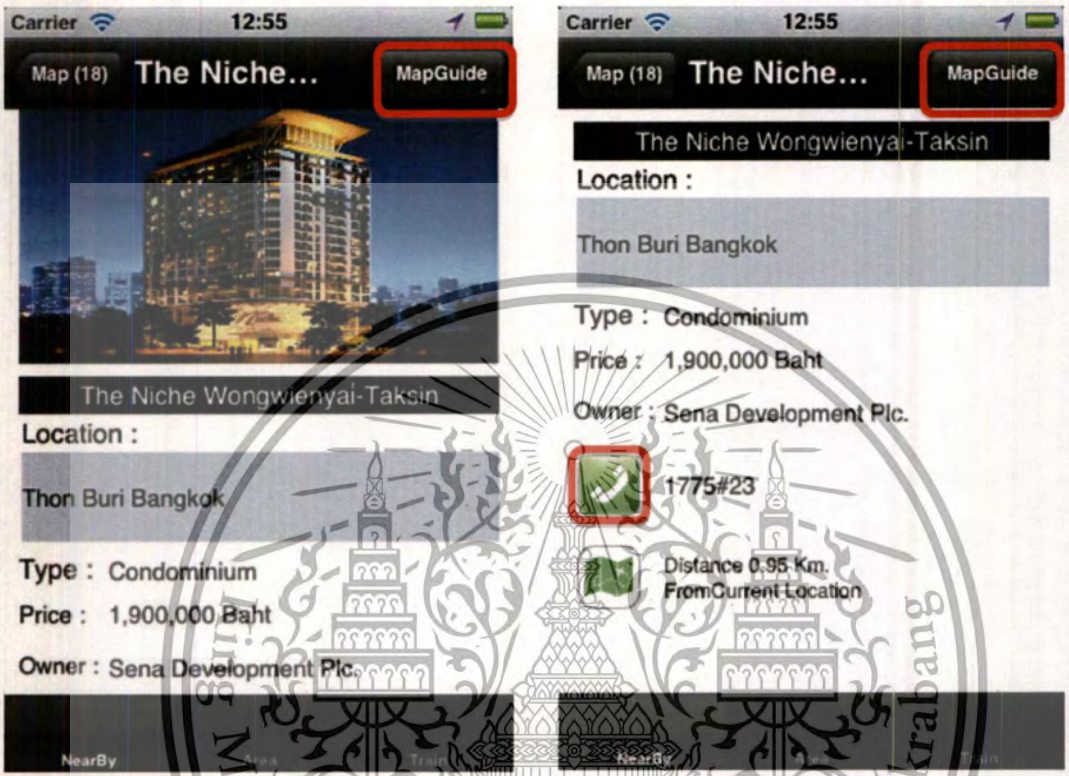


Figure 4.20 Accommodation information page from map view

Figure 4.19 show the information about each accommodation that the user chooses from the map view page. This page will show more information including picture, location, type, and price. The user can go back to the map view page to choose other accommodations or can go to the map guide page to get direction from his current position to the selected accommodation as shown in Figure 4.20

4.2.8 Map guide page

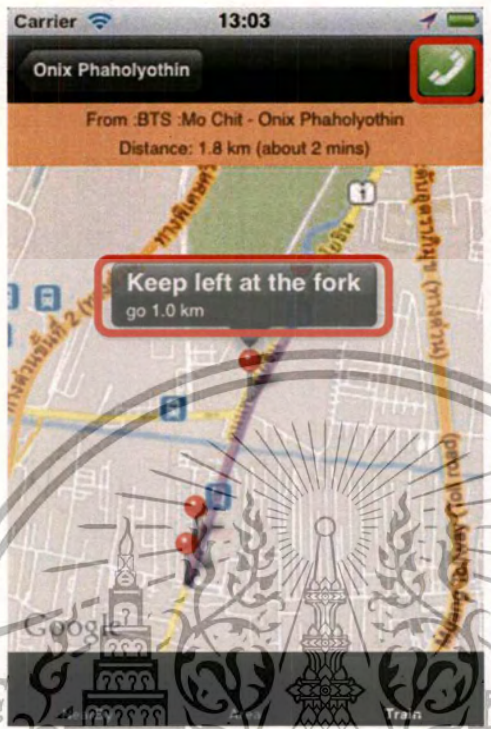


Figure 4.21 Map guide page

Figure 4.20 shows the path from the user current position to the selected house which the application provides more than one path so that the user can choose which path that he wants. The map also provides the description for each step. In this page, the user can make a phone call to the sale office directly by touching phone button.

```
-(id)initWithPlace:(Place *)__place fromMode:(int)__mode
searchMode:(int)__searchMode Lat:(NSString *)__Lat Lon:(NSString *)__Lon
fromPoint:(NSString *)__point{
    self = [super initWithNibName:@"MapGuideViewController" bundle:nil];
    if (self) {
        self.place =  __place ;
        self.lat = __Lat;
        self.lon = __Lon;
        self.searchMode = __searchMode;
        self.fromPoint =__point;
    }
    return self;
}
```

Figure 4.22 A method for getting data from the class model

Figure 4.21 shows a method the the mapguide uses to receive data from the detail page.

```
-(IBAction)showTel:(id)sender{

    NSString *resultString = [self.place.telNo stringByTrimmingCharactersInSet:
    [NSCharacterSet whitespaceAndNewlineCharacterSet]];

        if (![resultString isEqualToString:@""] ) {
        resultString = [resultString stringByReplacingOccurrencesOfString:@"-"
        withString:@""];
        resultString = [resultString stringByReplacingOccurrencesOfString:@" "
        withString:@""];
        resultString = [resultString stringByReplacingOccurrencesOfString:@","
        withString:@"#"];

        NSString *theString = resultString;
        NSString *substring = @"#";
        NSRange range = [theString rangeOfString:substring];

        if (range.length>0 && range.location>0) {
            resultString = [resultString substringToIndex:range.location];
        }
        NSString *telno = [NSString stringWithFormat:@"tel://%@", resultString];
        [[UIApplication sharedApplication] openURL:[NSURL URLWithString:telno]];
    }
}
```

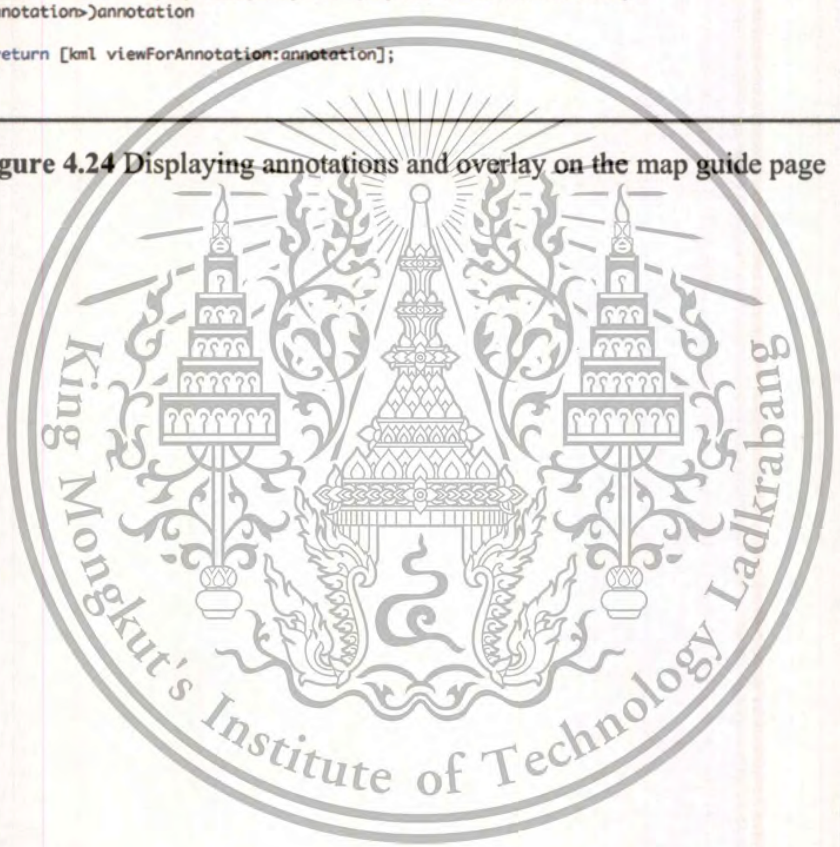
Figure 4.23 A method to make a phone call

Figure 4.22 shows a method to make a phone call to the sale office. Line 16 to 23 show how to edit the data (telephone number) retrieved from the database and use the number to call.

Figure 4.23 show after the application receives the data from the Google Map API, these two methods are called to display pins and route paths on the map guide.

```
32
33      - (MKOverlayView *)mapView:(MKMapView *)mapView viewForOverlay:(id
        <MKOverlay>)overlay
34      {
35          return [kml viewForOverlay:overlay];
36      }
37
38      - (MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id
        <MKAnnotation>)annotation
39      {
40          return [kml viewForAnnotation:annotation];
41      }
42
```

Figure 4.24 Displaying annotations and overlay on the map guide page



Chapter 5

Conclusion

5.1 Result

House Finder application was developed for iOS device's users (iPhone and iPad) to help them finding new accommodations (houses and condominiums) in Bangkok and its suburbs. The application provides 3 different searching types: Search by Nearby, Search by Area and Search by Train. Search by Nearby will search for accommodations around the phone's current location within 2-10 kilometers depending on the user's choice. Search by Area allows the users to choose the area (province and district) from the dropdown lists to search for accommodations within 10 kilometer in that area. Search by Train allows the user to search for accommodations located near the selected train station, which is chosen from the list of MRT, BTS and airport links (APL) station. The result will be displayed on the screen in two views, the list view and the map view. Application will provide basic information of each house for user to make decision easier and user can call to the house or condominium sale office from application directly to get more information. The short description of each accommodation result will be shown in the list view item which the user can see more detail by touching each item. The map view shows the searching result (accommodation's location) by placing the symbol of each accommodation on its position. The user can further press the symbol for more accommodation's detail and the direction to its place (map guide). The application also allows the user to make a phone call directly to the sale offices by just simply click at the phone icon in the detail and the map guide pages. The application was developed

on Xcode, used SQLite for the database and used soap protocol to communicate with the *Home Buyer Guide*'s web service. For the map, Google API is used.

5.2 Problems and Solutions

1. Because we had never used tools and programming languages for developing iOS application before, it took us quite a lot of time to learn. Also, synchronizing application from the MacBook Pro to iPhones or iPads tooks many steps (see Appendix D).
2. The information that we use in this application, we asked from the external real estate companies (*Home Buyer Guide)*. However, this information was in Thai and some information was not complete. Hence, we had to check manually to discard incomplete records before translating into English and uploaded to the company's server. This is quite time consuming.
3. The problem occurred when we used Google Map API together with Xcode, since it used the new library that we never used before so we needed to learn more about Google Map API and Map kit in Xcode.

5.3 Limitations of the project

1. Users have to go online while using the application since we obtain the information from the web service.
2. Application can run only on iOS devices.

3. Our application obtains data from the company's web service, which allows us to access only some services, and provides only some specific types of services. We cannot add any more functions to the application since the services are limited.
4. Our application can only search for accommodations in Bangkok, Nonthaburi, Phatumthani and Samuthprakarn.

5.4 Development in the future

1. Application should provide a recommended list for the popular and the newest places.
2. Application should provide a recommended list for the users appropriated with their profiles: salaries, type of houses, prices etc.
3. Application should have a function to allow users to rate the house so that this value can be used in evaluating the popular places.
4. Application should update news about the real estate project.

References

[1] [Online] http://en.wikipedia.org/wiki/IOS

[2] [Online] https://developer.apple.com/xcode/

[3] [Online] http://en.wikipedia.org/wiki/LLVM

[4] [Online] http://lldb.llvm.org/

[5] [Online] http://en.wikipedia.org/wiki/Google_Maps

[6] [Online] http://en.wikipedia.org/wiki/JavaScript

[7] [Online] http://en.wikipedia.org/wiki/greasemonkey

[8] [Online] http://www.google.com/earth/outreach/tutorials/annotate.html

[9] [Online] https://developers.google.com/maps/

[10] [Online] http://developer.yahoo.com/maps/

[11] [Online] http://www.microsoft.com/maps/

[12] [Online] http://www.mapquest.com/

[13] [Online] http://openlayers.org/

[14] [Online] http://www.spatial.cs.umn.edu/Courses/Fall11/8715/papers/IM7_steiniger.pdf

[15] [Online] http://www.sqlite.org/about.html

[16] [Online] http://en.wikipedia.org/wiki/Objective-C

[17] [Online] http://en.wikipedia.org/wiki/Xml

[18] [Online] http://en.wikipedia.org/wiki/Keyhole_Markup_Language

[19] [Online] http://en.wikipedia.org/wiki/Marble_(KDE)

[20] [Online] http://en.wikipedia.org/wiki/Geography_Markup_Language

[21] Virrantaus, K., Markkula, J., Garmash, A., Terziyan, Y.V., 2001. Developing GIS-Supported Location-Based Services. In: Proc. of WGIS'2001 – First International Workshop on Web Geographical In-formation Systems., 423–432

Appendices

Appendix A

Xcode and SQLite Plugins Installation

How to install Xcode and iOS SDK

1. Download Xcode from Mac App Store.

2. Double click on Xcode and iOS SDK.mpkg to install the program (see Figure A.1).

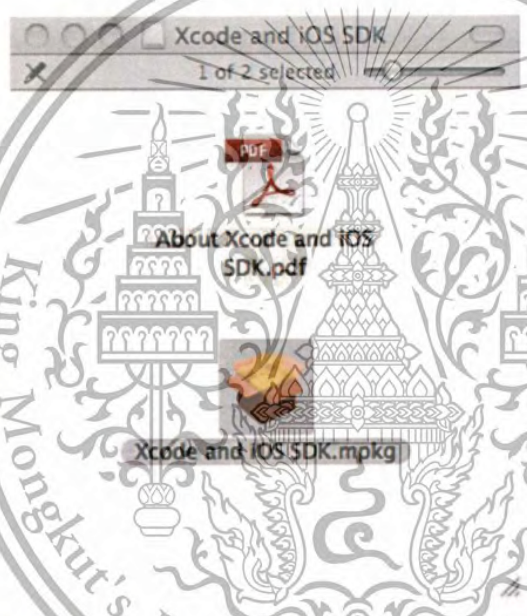


Figure A.1 Xcode and iOS SDK.mpkg installation

3. Click **Continue button**, to start installing the Xcode (see Figure A.2).

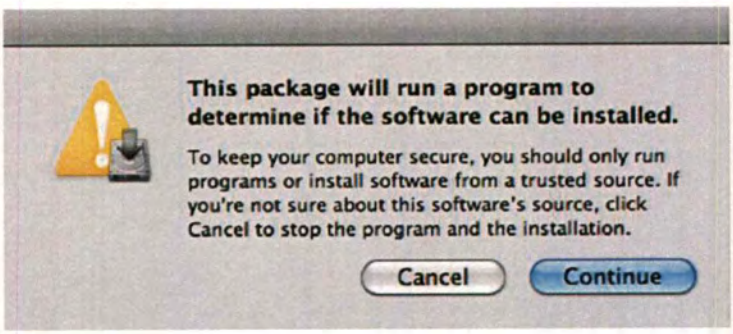


Figure A.2 Security alert pop up window

4. Click **Continue button**, to continue installing the Xcode (see Figure A.3).

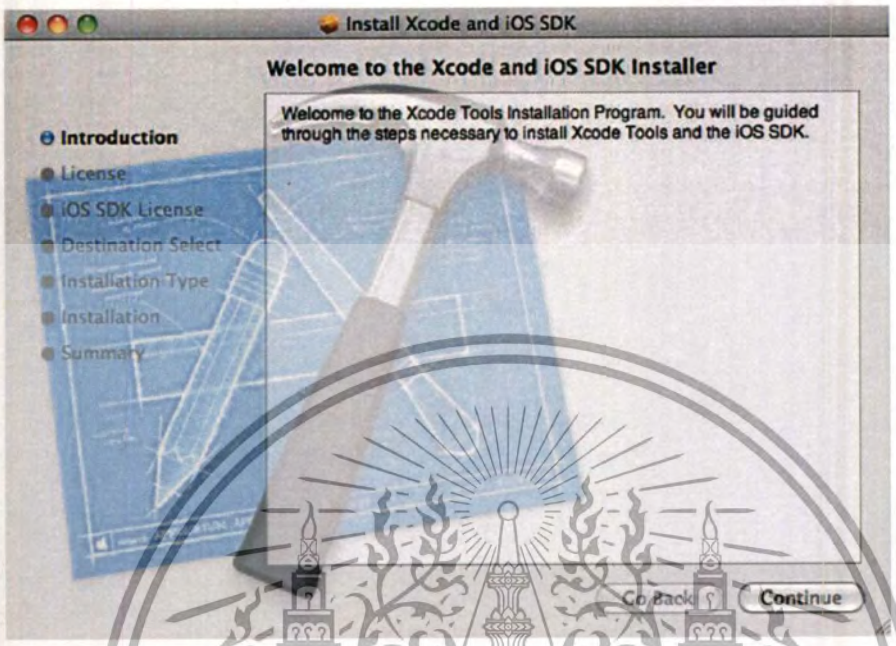


Figure A.3 Introduction page of Xcode installing window

5. Click **Continue button**, to accept the software License Agreement (see Figure A.4).

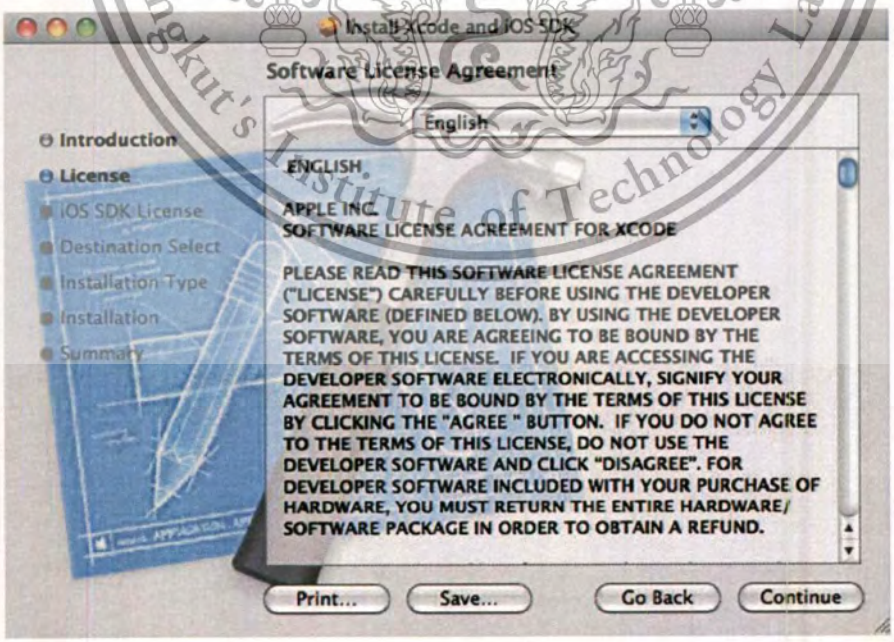


Figure A.4 Software License Agreement page of Xcode installing

6. There is a pop up window of Xcode License Agreement, Click **Agree button** (see Figure A.5).

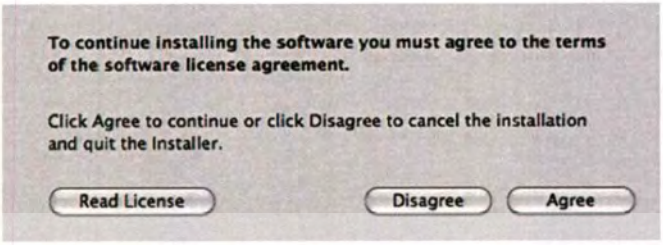


Figure A.5 Xcode software license agreement pop-up window

7. A pop up window of iOS SDK License Agreement will be displayed, click **Agree button** (see Figure A.6).

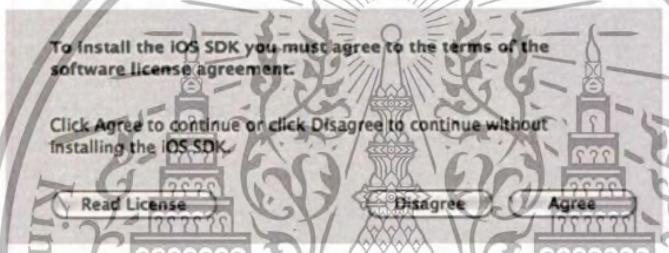


Figure A.6 iOS SDK software license agreement pop-up

8. Check at the check box of the installation type that you would like to install then click **Continue button** (see Figure A.7).

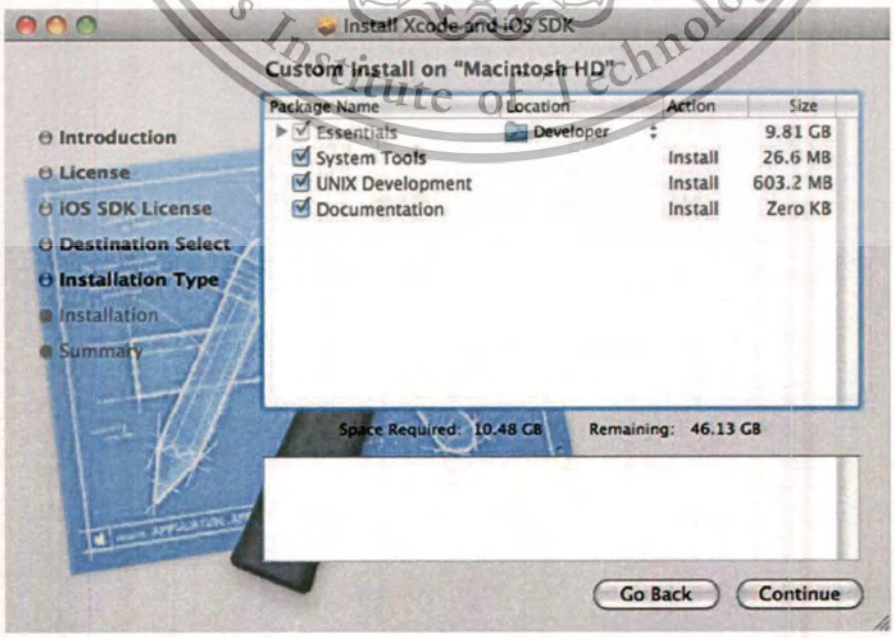


Figure A.7 Installation type page at Xcode installation window

9. Wait for 20 minutes, Xcode will be installed automatically (see Figure A.8).

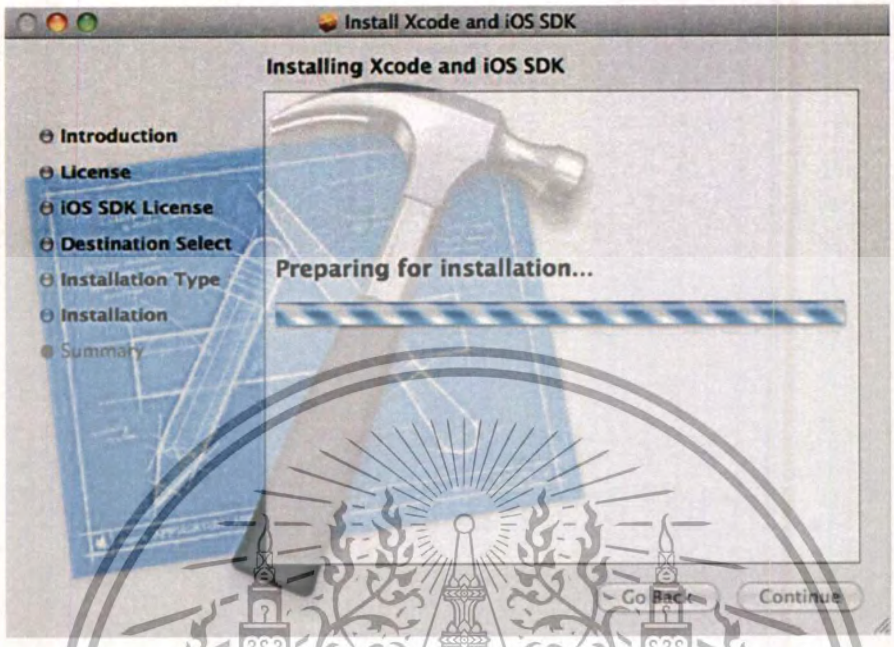


Figure A.8 Installation window while working

10. After Xcode installation is successful, click on **Close button** to launch the Xcode (see Figure A.9).

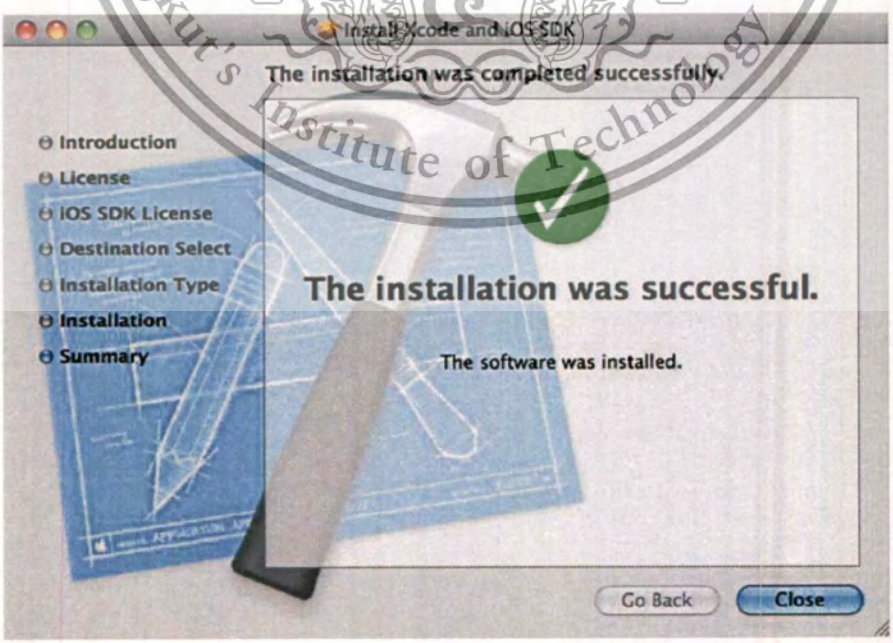


Figure A.9 Installation was complete

How to install SQLite plugins on Firefox

1. Download SQLite manager from https://addons.mozilla.org/en-US/firefox /addon /sqlite-manager/

2. Click + Add to Firefox to install SQLite manager to Firefox (see Figure A.10)

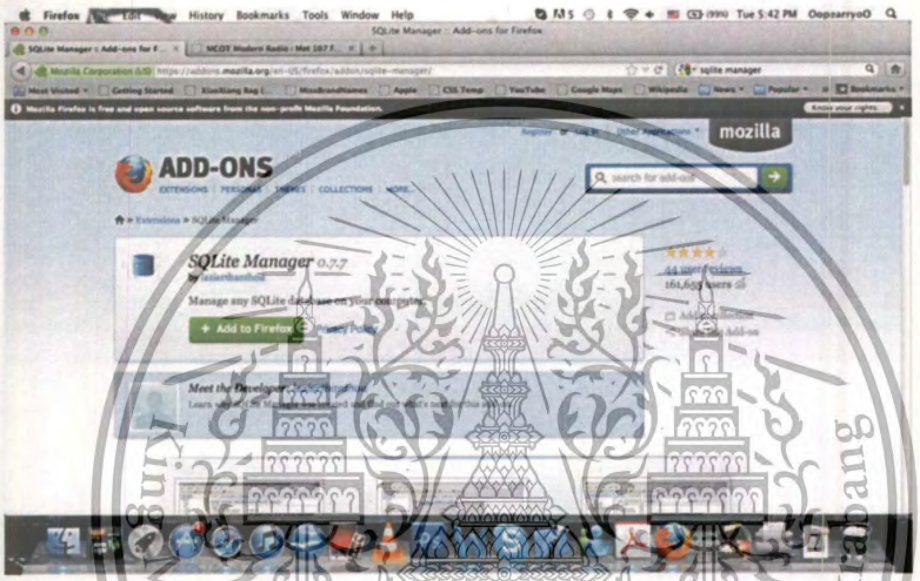


Figure A.10 Firefox add-ons page

3. Click **Install Now** button (see Figure A.11)

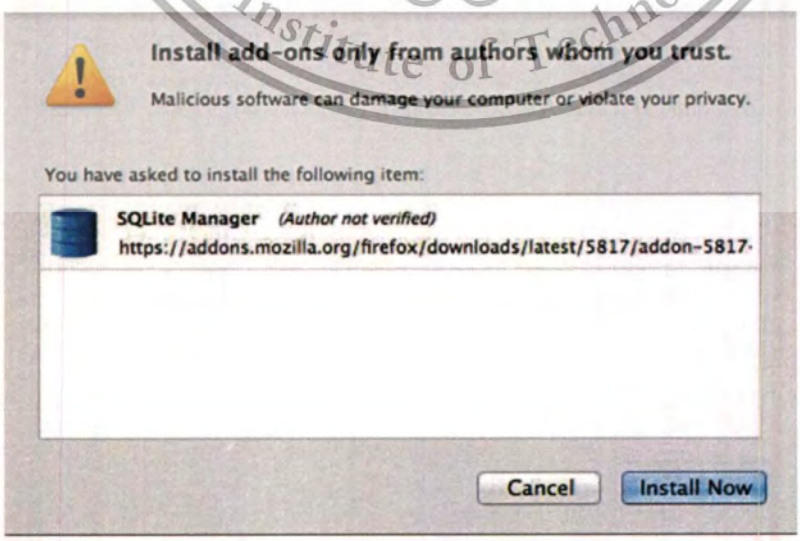


Figure A.11 Security alert window

4. Click **Restart Now** button, to restart Firefox (see Figure A.12)

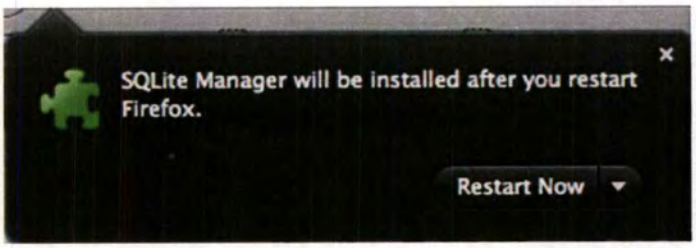


Figure A.12 Information pop-up window

5. Click on **Tool** at the toolbar, choose **SQLite Manager** (see Figure A.13)

Figure A.13 Firefox browser

6. SQLite manager (editor) are shown on the screen (see Figure A.14)

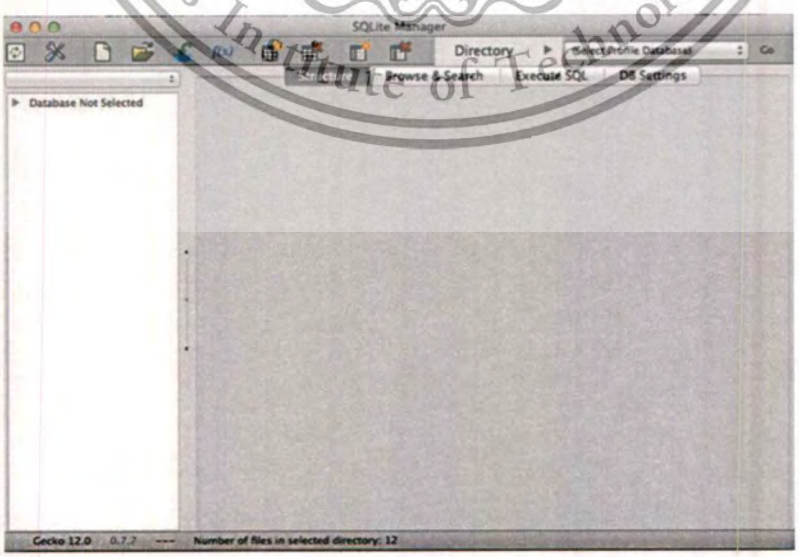


Figure A.14 SQLite manager application

Appendix B.

User Manual

User Manual for House Finder Application

The House Finder Application aims at providing users the collections of accommodations available for sale. In just a few clicks, users can find a list of available accommodations that they are looking for. In addition, the application also provides a map, which can navigate them to their interested accommodations from either, their current location, a reference point of area that they are looking for or from the train station. Also all required information about each accommodation such as (price, picture, construction's company, phone number, etc.) would also be provided.

At this stage, our application only provides the accommodation information in Bangkok, Nonthaburi and Patum thani. There are six types of accommodations available to search: single-house, twin-house, town-home, condominium, town house and commercial building. For the location searching options, depending on the user requirements, our application provides three alternative location search options: **Search Nearby**, **Search by Area** and **Search by Train**. Search Nearby provides a list and a map of the accommodations situated nearby the user current location. **Search by Area** provides a list and a map of the accommodations, which situated near the area that the users select. Our application uses the district name to specify the area. At this stage, we only provide districts in Bangkok. **Search by Train** provides a list of accommodations located near the train stations (either BTS or MRT or Airport link).

When you launch our House Finder application, the main screen as shown in Figure B.1 will appear. We note that the default page of the main screen is the **Search Nearby** page.

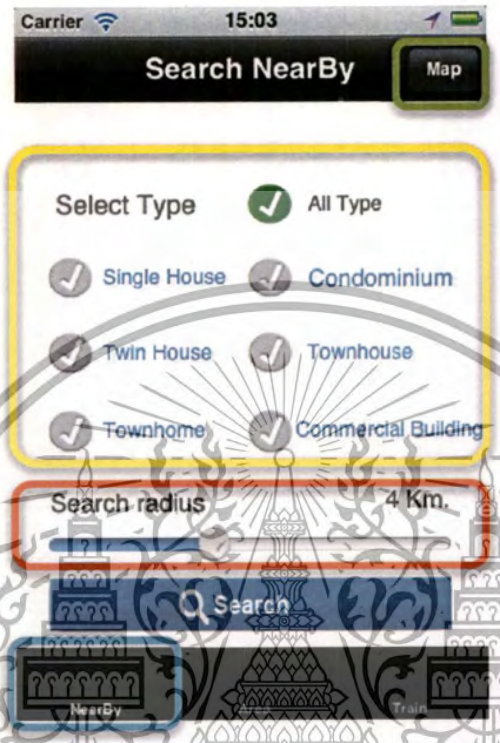


Figure B.1 Search by nearby page

A.1. Search Nearby

Steps to use the application in the **Search Nearby** are listed below:

1. On the **Search Nearby** page, touch on the accommodation type you want to search. You can choose more than one types or you can choose **All Type** if you want to search for all (see Figure B.1-yellow section).
2. Scroll left and right to adjust the searching radius (see Figure B.1-orange section). Note that the maximum radius is 4 kilometers.
3. Touch on 

, then you will see the searching result in the list view as shown in Figure B.2.

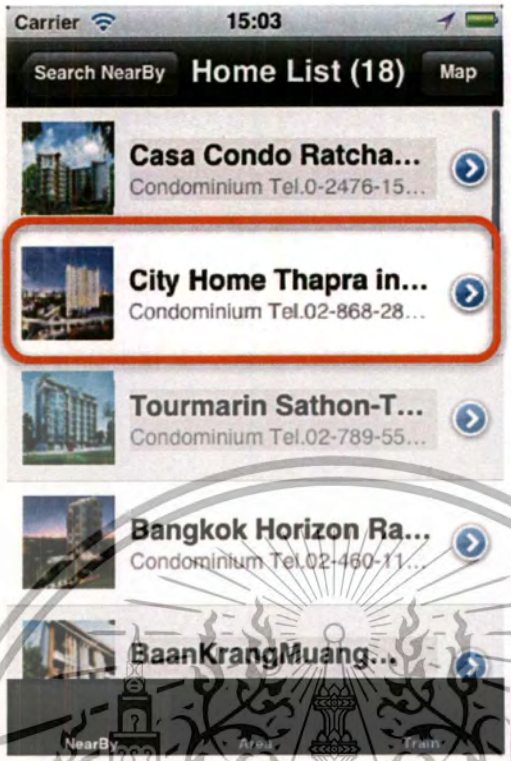


Figure B.2 Search result on list view

4. In Figure B.2,

   4.1 If you want to see more detail of each accommodation item, touch on its item which then the accommodation full information will be displayed. An example of the full information page is shown in Figure B.6.

   4.2 If you want to go back to the previous page, touch Search NearBy button.

   4.3 If you want to see the result in the map view touch on Map You will see the search result in the map view as in Figure B.3.

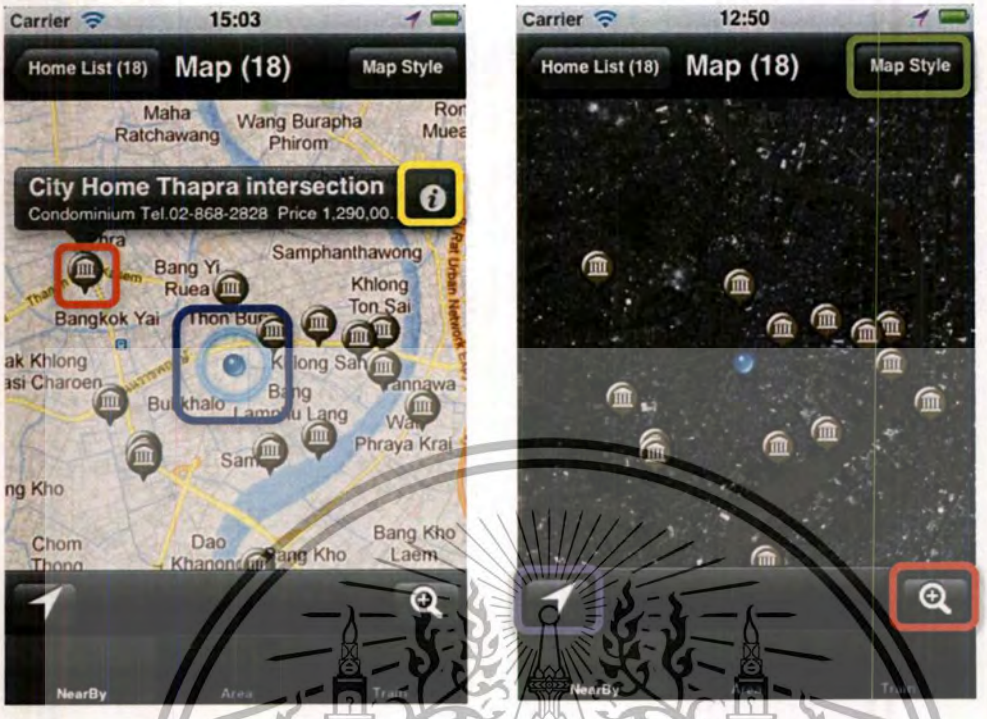


Figure B.3 Search result on map view in standard and satellite modes

5. In Figure B.3, the map page in the standard mode is shown. Your location will be marked with the blue circle at the center of this page (see blue section). A symbol represents the location of the accommodation that satisfies the user search criteria (nearby in the specified radius and specified accommodation type) which if you touch it, the simple information about each place will be shown (orange section). For more information about each place touch on (Figure B.3) for its detail page as shown in Figure B.7
6. If you want to change from the standard map mode to the satellite map mode (Figure B.3-second figure), touch at Map Style on the upper right of page in Figure B.3 – first figure).
7. For the satellite map in Figure B.3, touch on when you want to see your location on the map (see purple section).

8. Touch on  to Zoom in.

A.2 Search by Area

As mentioned above, our application also allows you to search for the accommodations in your selected area.

1. First, touch on the **Area tab** on the bottom of the main menu page in Figure B.1 which the screen as in Figure B.4 (left) will appear.
2. Touch on **City** drop-down menu to select a province (Bangkok Nonthaburi or Patum Thani) that your required accommodations are located (see Figure B.4-orange section). You will notice that once you select the province, lists of that province's district will be loaded to the drop down box under @Area.
3. Touch on @Area menu to select the district that locates in the selected province (see Figure B.4-red section). Select Aumphur by touching at the down arrow to see more of the list (see Figure B.4)
4. Touch at the drop down list under **Home Type** to select the type of accommodation that you want to search as in Figure B.4

5. Similarly to **Search Nearby**, Search you want to see the result in the map view touch on. A map page similarly to one in Figure B.3 will be shown. Further information can be acquired the same way as the map view function on Search Nearby.

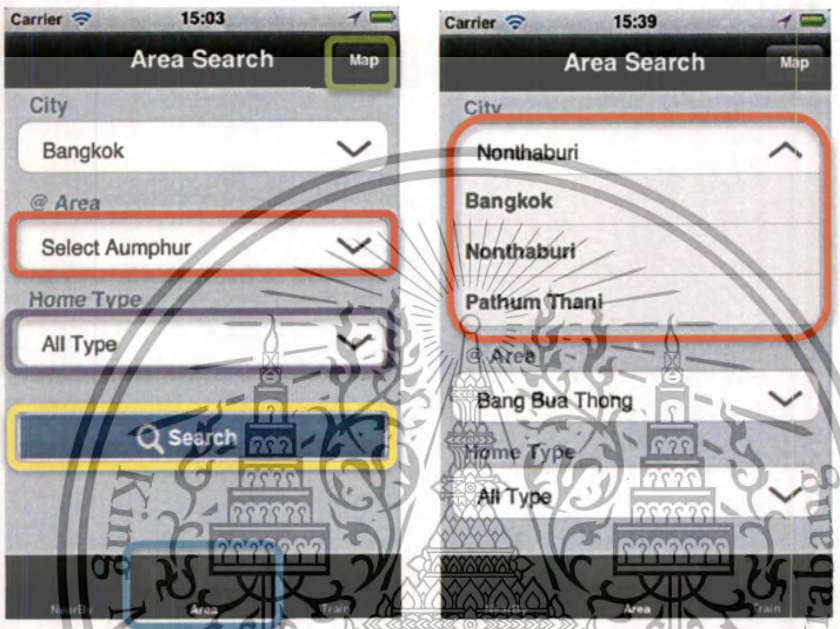


Figure B.4 Search by area page

A.3 Search by Train

Our application provides three types of train services: BTS, MRT and Airport link. In this type of search, the application will search for accommodations according to the station's location. Steps to **Search by Train** are listed as follows:

1. Touch on **Train tab** on the tab bar on the bottom right. Screen as in Figure B.5 (left) will appear.

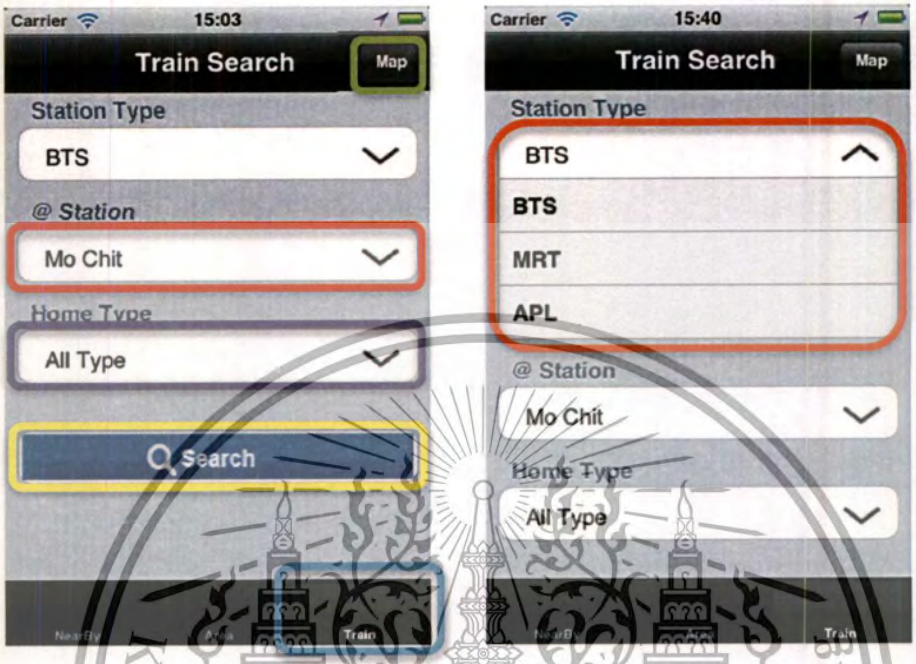


Figure B.5 Search by train station page

2. Touch on the box under **Station Type**. The drop-down menu to select the train service either BTS or MRT or APL (Airport link) as in Figure B.5 (right) will appear (see orange section).
3. Touch on the box under **@Station** menu to select the selected service's station (Figure B.5-red section).
4. Touch on the box **Home Type** to choose what kind of accommodation that you want (see Figure B.5-purple section).
5. Touch on Search to start searching. The searching result will appear in the list view the same way as **Search Nearby** as in Figure B.2

6. Similar to the Search Nearby, touch  to see the search result in the map view (see Figure B.3). For further search or use on the list view and the map view, the usages are the same as the Search Nearby (since Step 5).

A.4 Appearance and Usage of Detail pages and Map guide page

Since all search types eventually provide the similar result either for the list view or for the map view and the mapguide. We group them together and show their usages in this section. Not that highlighted tap on the tab bar in the bottom of every page shows the search type you are currently on.

A.4.1 Detail page from the list view

In the list view page of every search type, if you click the item in each list, the screen as in Figure B.6, which shows the full information of the selected accommodation, will appear. In this example, this accommodation is the result of **Search by Area.** Other buttons/taps, the usages are as explained below.

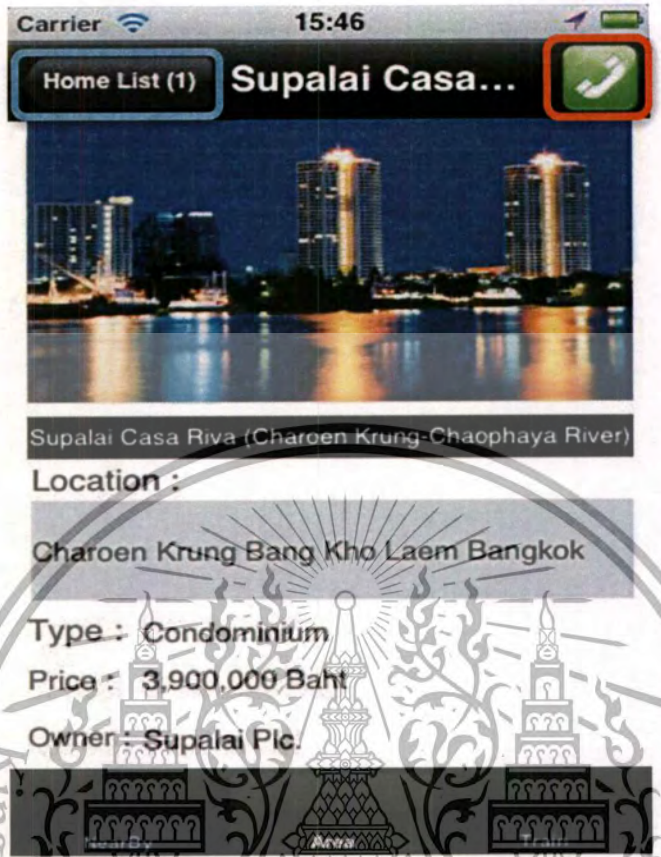
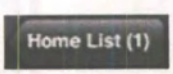


Figure B.6 House information page from list view

1. Touch on **call button** to make a phone call from the application directly (Figure B.6-orange section)
2. This page allows you to go back to the list page by touching 

button (Figure B.6-blue section)

A.4.2 Detail page from the map view

When you touch on 

button for more detail of the selected place on the map screen as in Figure B.3, the screen as in Figure B.7 will appear. In this example, this accommodation is the result of **Search by Nearby**.

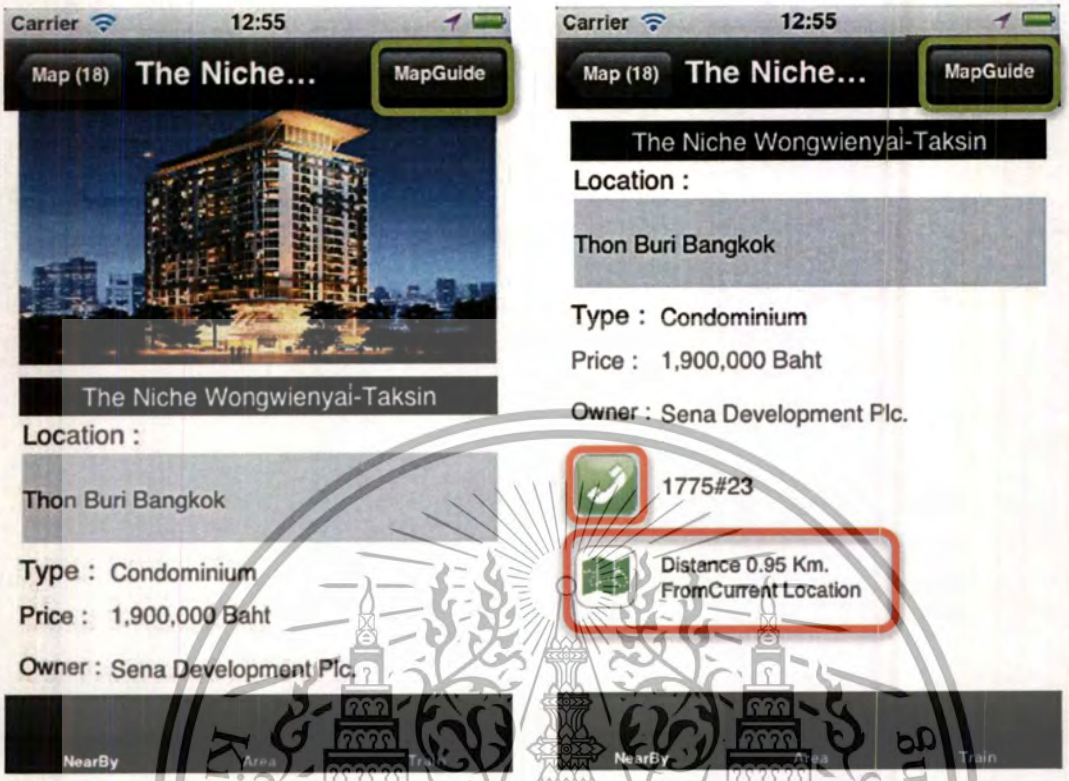


Figure B.7 House information page from map

Touch on **call button** to make a phone call from the application to the sale office directly (see Figure B.7- green section).

1. Below the **call button** (see Figure B.7-red section) there is information about distance between user current position and the selected house.
2. If you want to go back to the map view, touch Map (18) button to go back to the map view (see Figure B.3). The number in the Map(18) denotes the number of accommodation results on the result page.
3. Touch on MapGuide button to go to the map guide page, which will navigate you from your current location to your selected house (see Figure B.8).

A.4.3 Map Guide page

The map guide page will appear after you touch on the MapGuide s page will show you the direction from your current location **(Search Nearby)**or your selected train station **(Search by Train)** to your seleced accommodation. In this example (Figure B.8) , it is the map from the Mochit BTS station. The usage of the map is as follows:

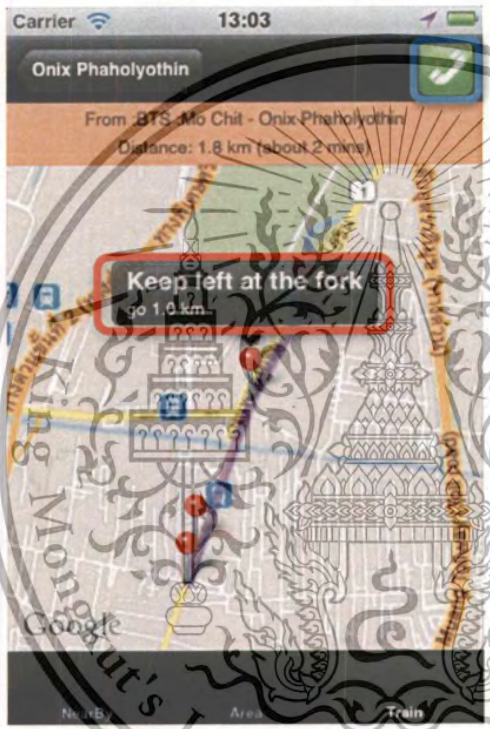
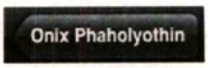


Figure B.8 Map guide page

1. Touch on  to see the direction of each step (see Figure B.8-red section). Note that the red pin denotes the description on each step.
2. Touch on **call button** to make a phone call to the sale office.(see Figure B.8-blue section)

3. Touch on 

button to go back to the detail page (see Figure B.7)

Appendix C.

Yearbook: User Manual

User Manual for Yearbook Application

The Yearbook application provides 3 main functions; the first function is for showing the information about each person who has information in our database. The second function is a call function which allows the user to make a phone call to any contact in this application. The last function is a function that provides a facebook button which pressing this function will show the facebook page of that particular person.

When you launch the Yearbook application, the home screen as in Figure C.1 will appear.

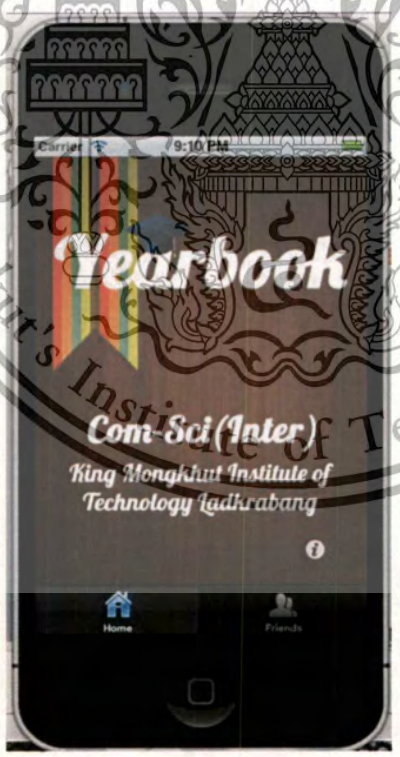


Figure C.1 Home screen of Yearbook Application

1. Touch on  button to see the Information page which shows a picture of students in our class (see Figure C.2).
2. Touch on  ab to go to the Contact list page (see Figure C.3).

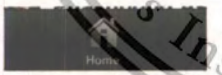
Information Page

Figure C.2 Information page

1. Touch on  button to go back to the homepage as in Figure C.1.

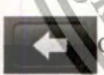
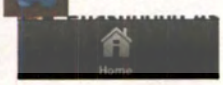
Contact List Page

Figure C.3 Contact List page

1. Touch on each row to see more information about each person as shown in Figure C.4
2. Touch on  tab to go back to home page (see Figure C.1).

Detail Page

Figure C.4 Detail page

1. Touch on  or  button to go back to the contact list page (see Figure C.3)
2. Touch on  button to use call function which application provide for user to make a phone call to each person directly from this application.
3. Touch on  button to link to the facebook page (see Figure C.5)
4. Touch on  button to go back to homepage (see figure C.1).

Facebook Page

Figure C.5 Facebook page

1. Touch on  to go back to the detail page (see Figure C.4)
2. Touch on  button to go back to homepage (see Figure C.1).

Appendix D

Xcode Tutorial

How to create new project in Xcode

1. Launch Xcode program
2. Create new project by Click File ➔ New ➔ Project
3. Choose **Empty Application** then click **next button** (see Figure D.1)

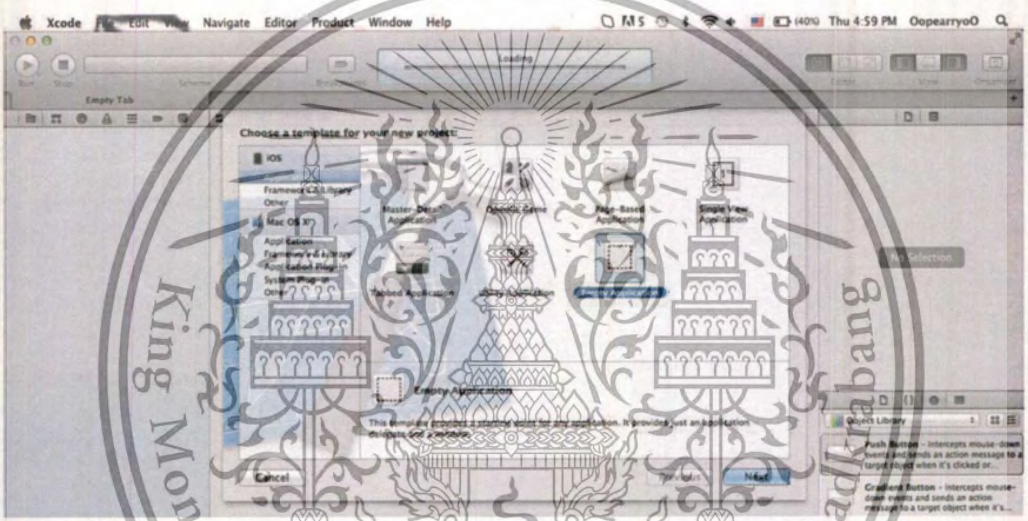


Figure D.1 Template for new project window

4. Name the new project at filed **Product Name** then click **Next button**. (see Figure D.2)

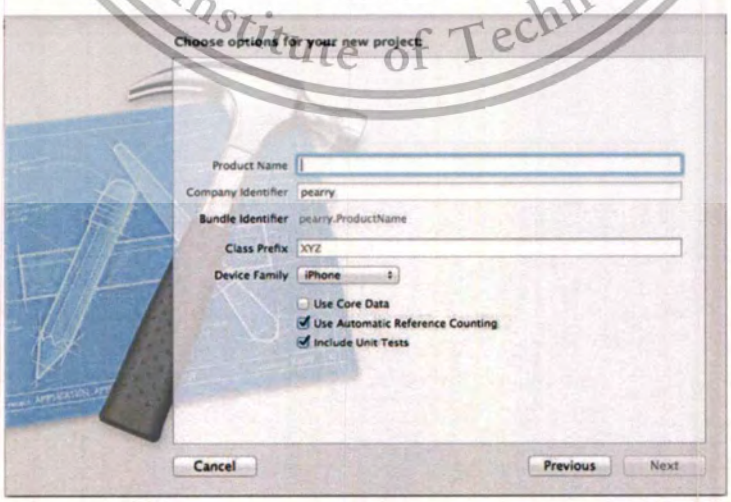


Figure D.2 Option for new project window

5. Choose the location that user need to save the new project. (see Figure D.3)

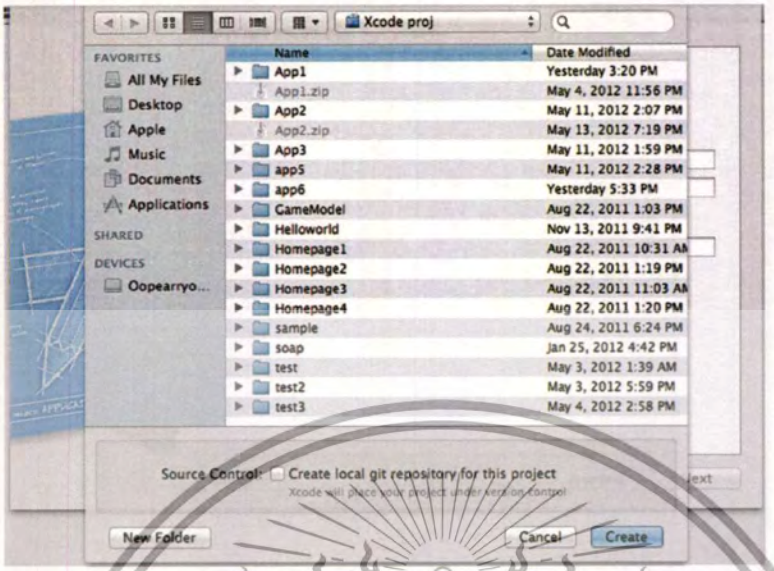


Figure D.3 File directory

6. Figure D.4 is the home screen after create new project file, click on the icon which represent each orientation to set the device orientation (see Figure D.4- red section)

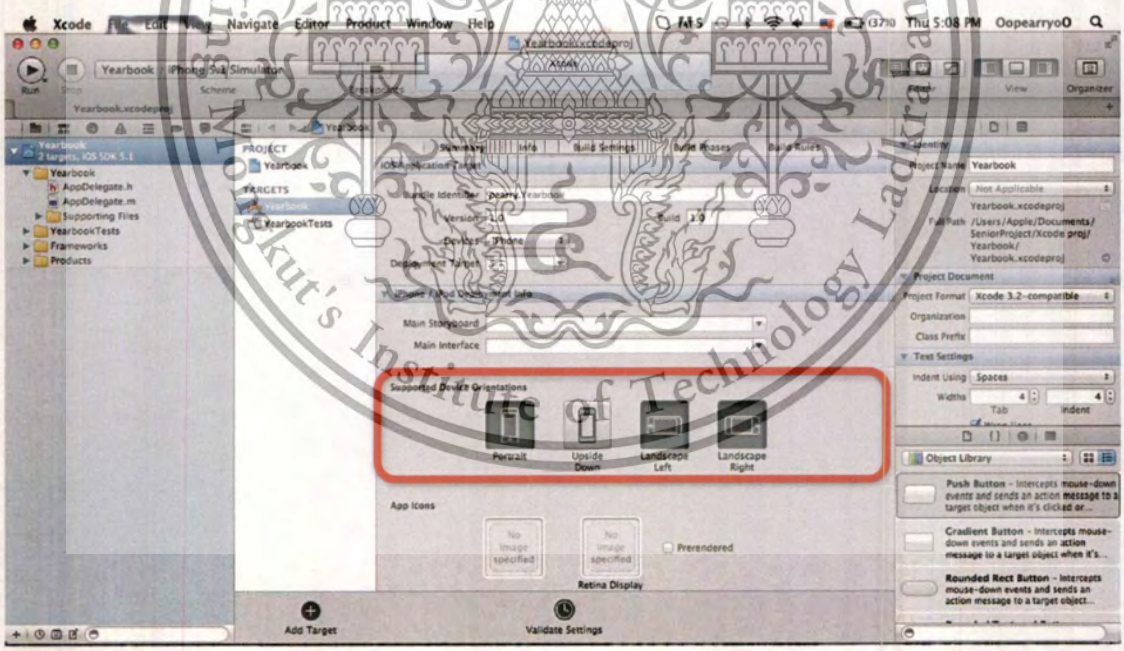


Figure D.4 Xcode home screen

7. Click File ➔ New ➔ File (see Figure D.5-orange section), to create new file inside this project.

8. Click on **Interface**, choose **Empty** then click **Next button** (see Figure D.6 – D.7)

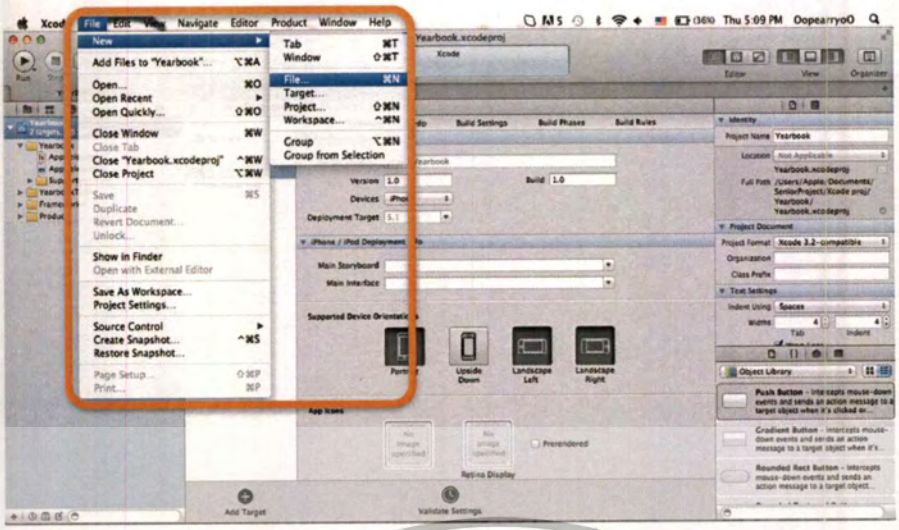


Figure D.5 Create new file command

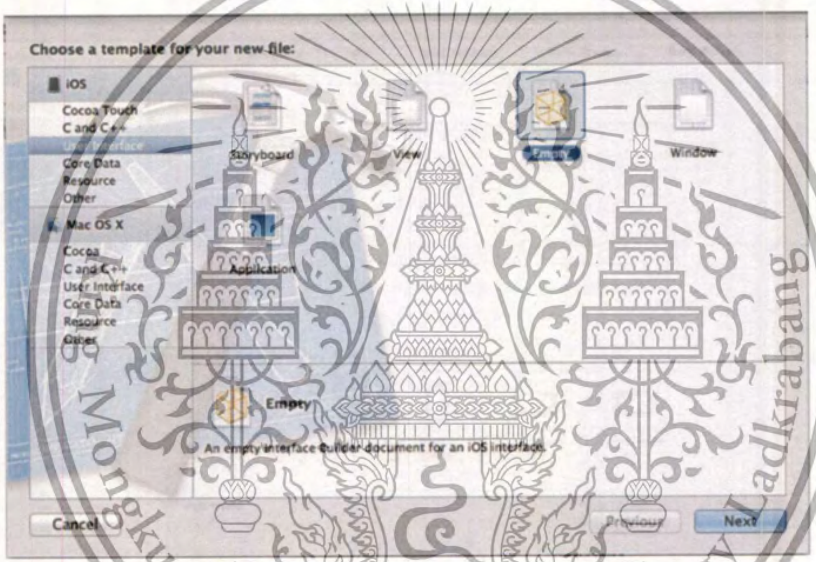


Figure D.6 Template for new file window

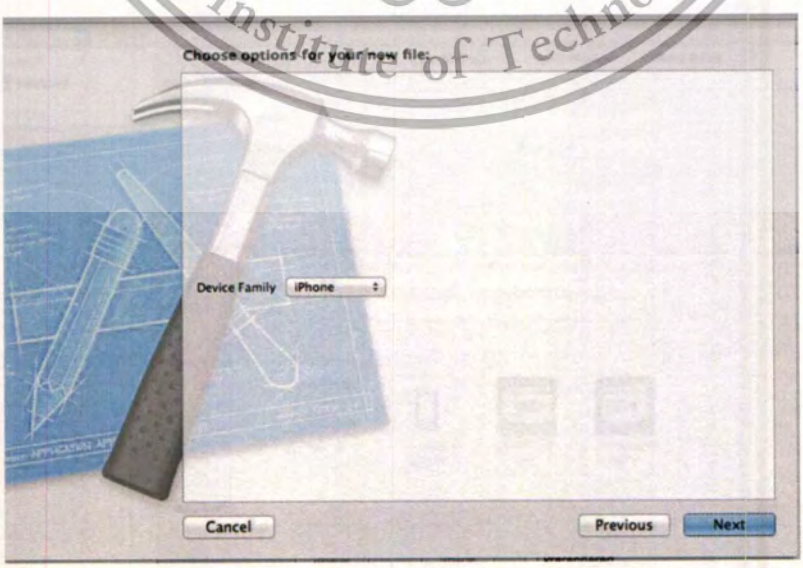


Figure D.7 Option for new file window

9. Name this file *"MainWindow"*, then choose the location that user need to save this file (see Figure D.8)

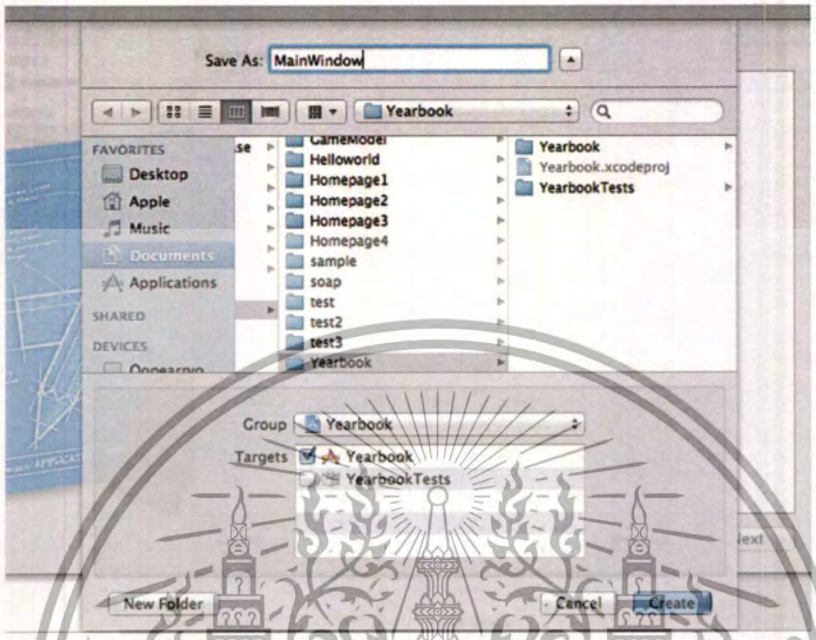


Figure D.8 File directory

10. *MainWindow.XIB* interface builder window (see Figure D.9)
11. Create *object* and *window* (see Figure D.10); by drag icon at the bottom right panel to the white grid space (see Figure D.9-yellow section).

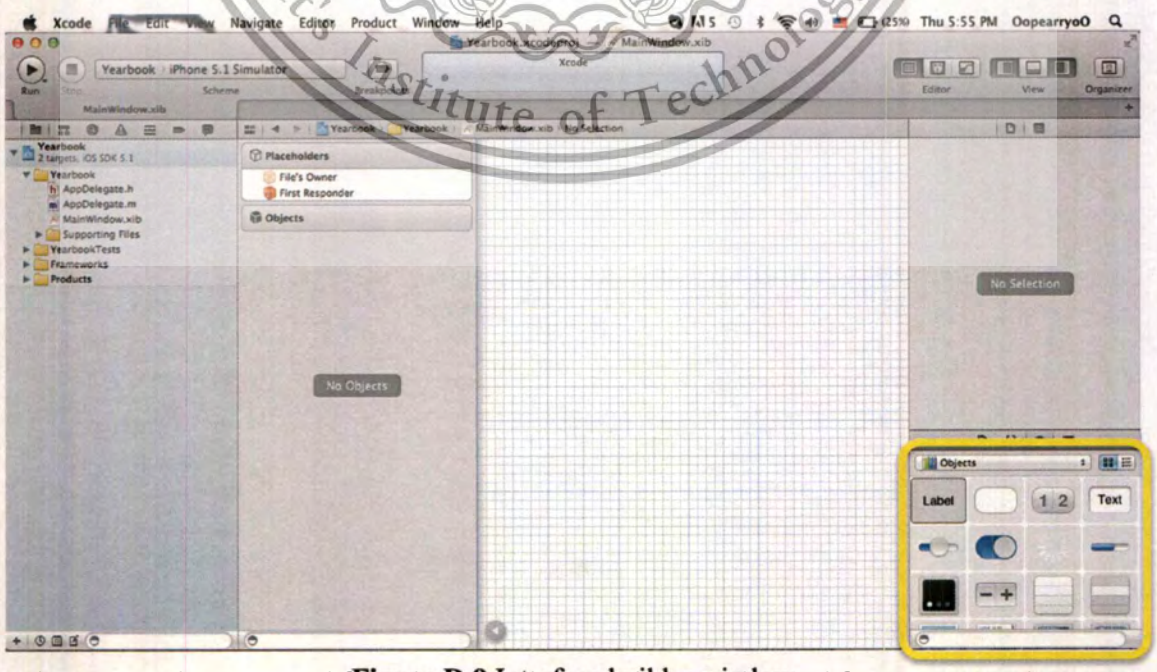


Figure D.9 Interface builder window

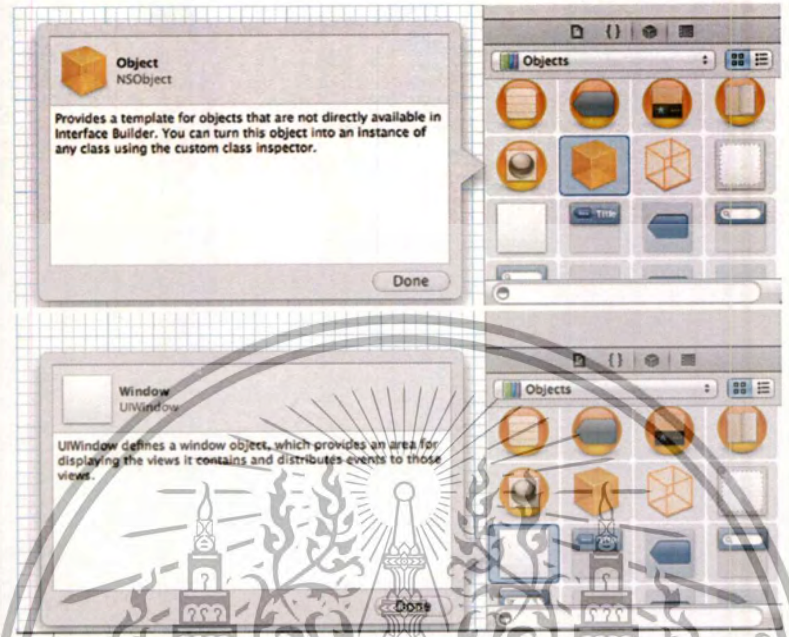


Figure D.10 *Object* and *Window* at the bottom right panel

12. After drag two objects to the screen as show in Figure D.11

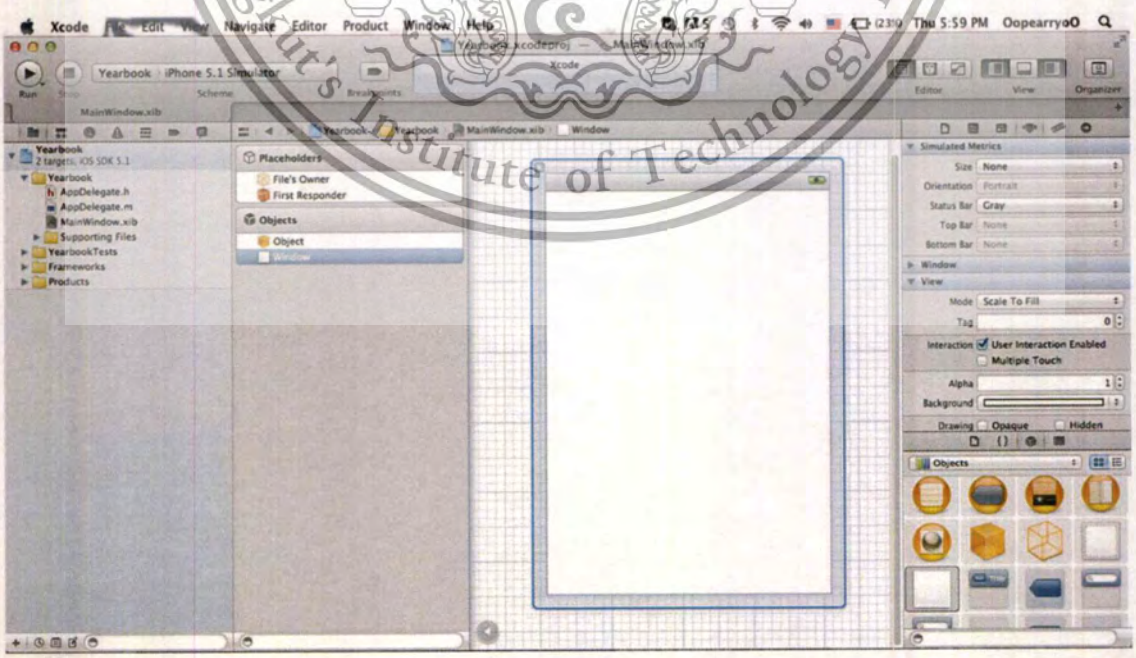


Figure D.11 Interface builder window

13. Create *Tab Bar Controller* and *Navigation Controller* (see Figure D.12)

14. After drag two objects to the screen as show in Figure D.13

Figure D.12 *Tab Bar Controller* and *Navigation Controller*

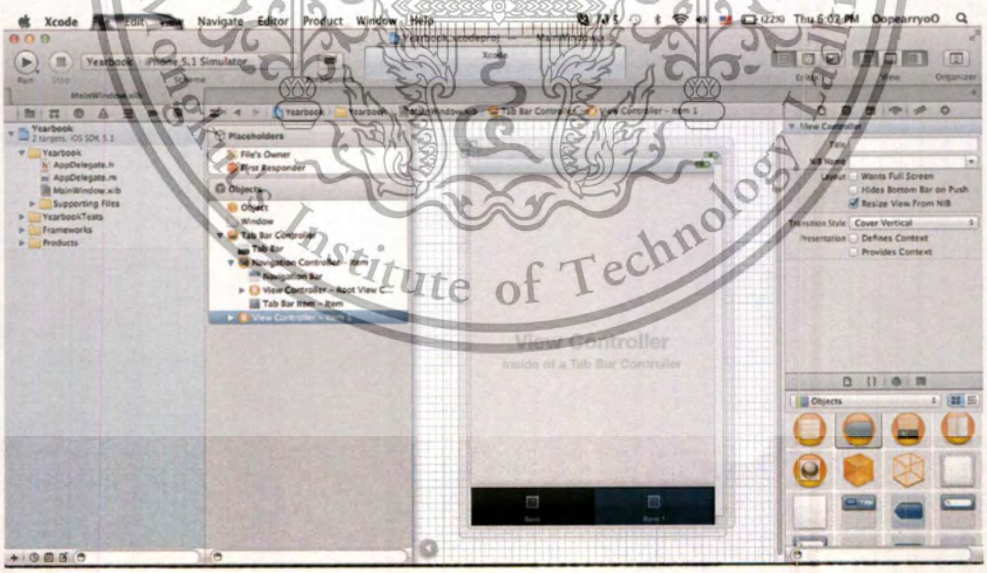


Figure D.13 Interface builder window

15. Go to AppDelegate.h (see Figure D.14), then type following the example (see Figure D.15)

Figure D.14 *AppDelegate.h* window

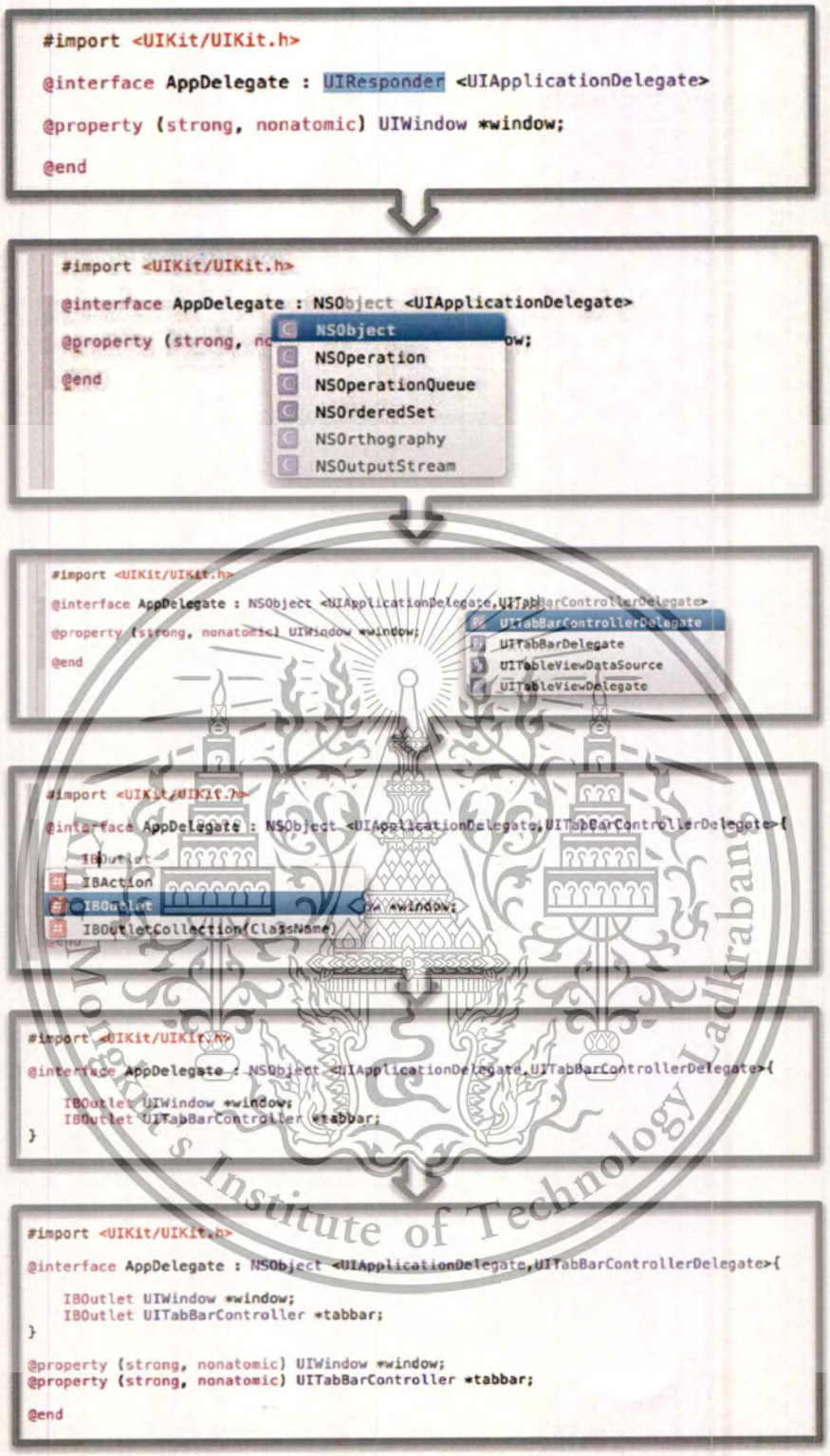


Figure D.15 Declare each variable in *AppDelegate.h*

16. Go to *AppDelegate.m* (see Figure D.16), then type following the example (see Figure D.17)

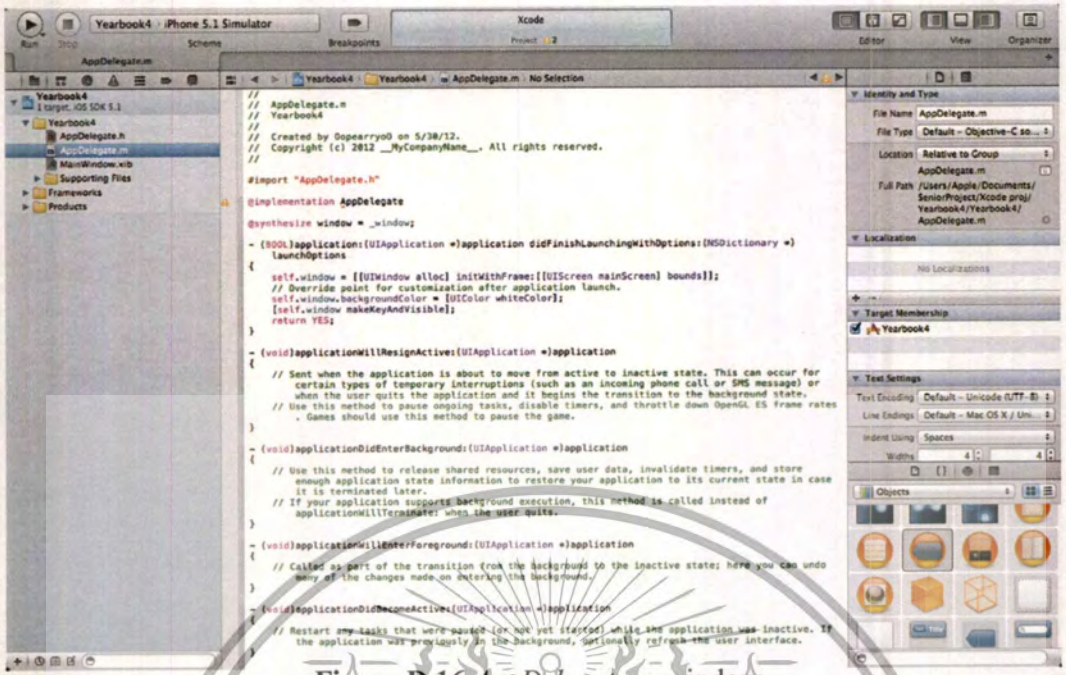


Figure D.16 *AppDelegate.m* window

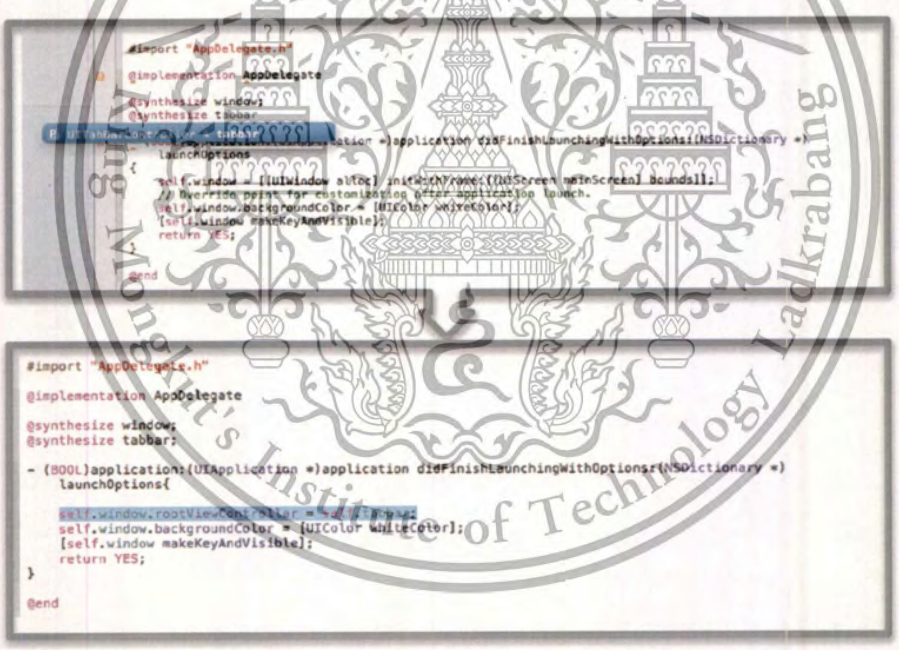


Figure D.17 declare each variable in *AppDelegate.m*

17. Go to *MainWindow.XIB*

18. click on ***File's Owner*** then click on Utilities ➔ identity inspector ➔ custom class ➔ type *UIApplication* (see Figure D.18-green section)

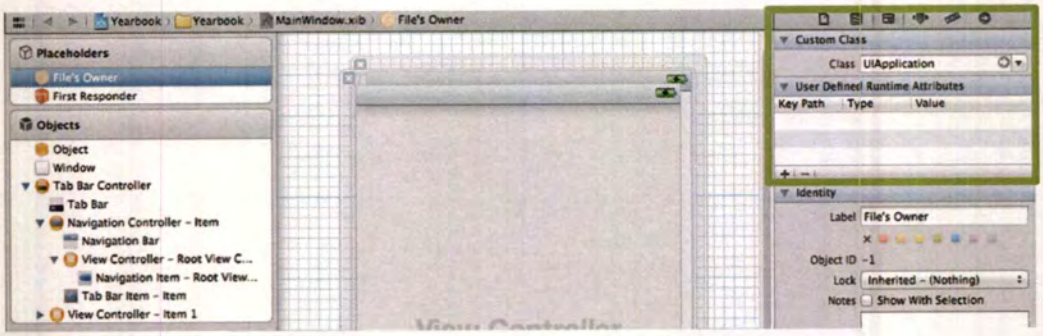


Figure D.18 *MainWindow.XIB* window File's

19. Click on *Object* then click on Utilities ➔ identity inspector ➔ custom class ➔ typing *AppDelegate* (see Figure D.19-orange section)

Figure D.19 *MainWindow.XIB* window object

20. Right click at *File's Owner* then drag *delegate* with *AppDelegate* (see Figure D.20)

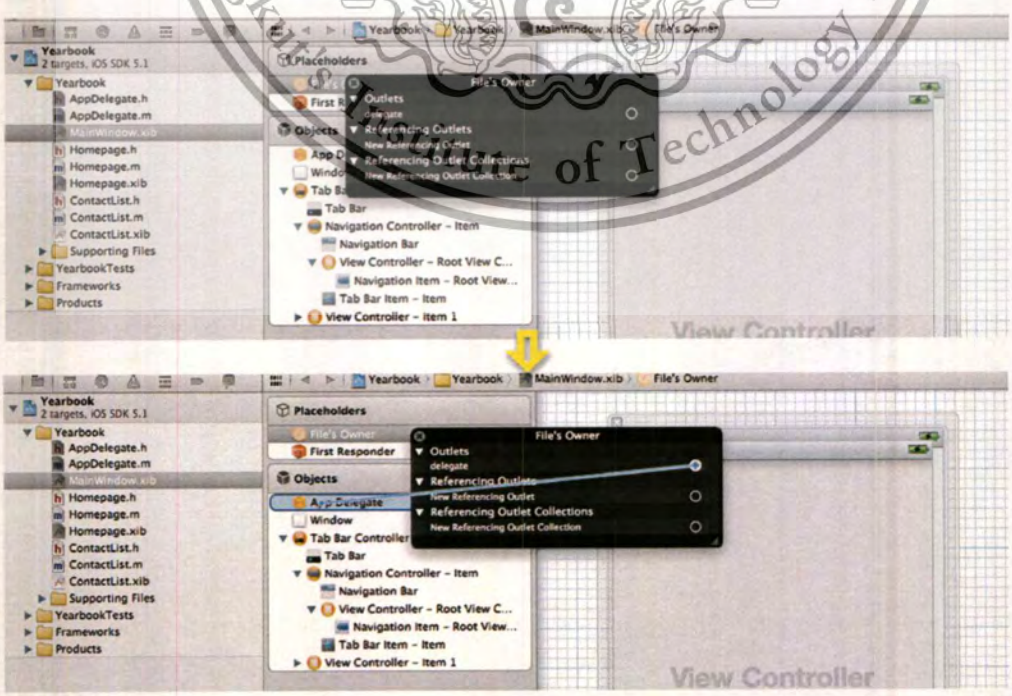


Figure D.20 File's Owner connection path at *MainWindow.XIB*

21. Right click at *AppDelegate* then drag *tabbar* with *Tab Bar Controller* and *window* with *Window* (see Figure D.21)

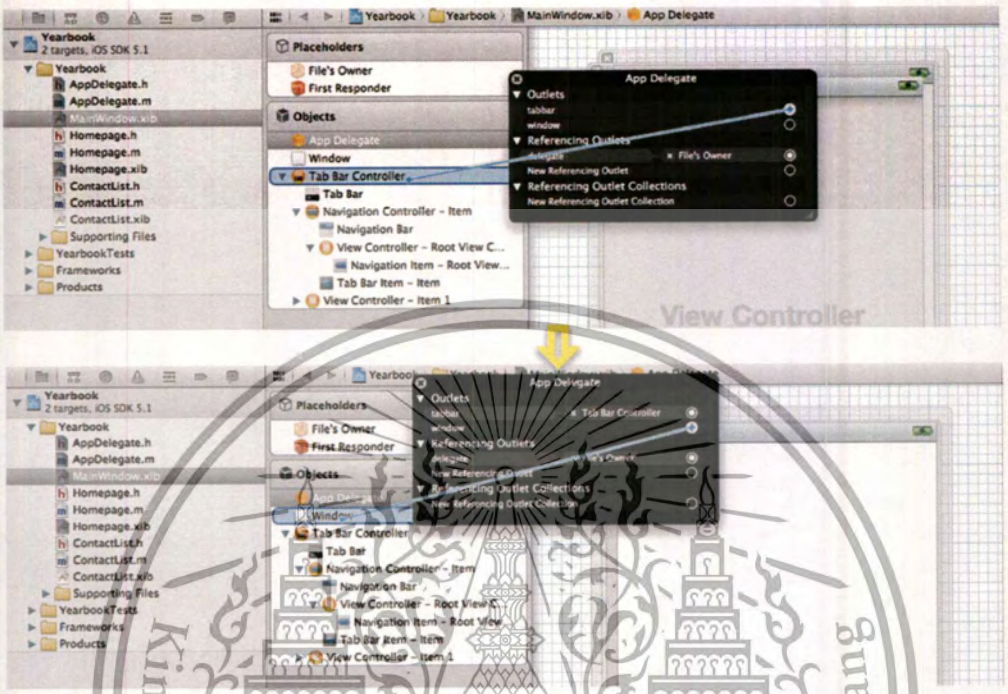


Figure D.21 *AppDelegate* connection path at *MainWindow.XIB*

22. Create new File, click on File ➔ New ➔ File

23. Click on *Cocoa Touch*, choose *Objective-C class*. Click **Next button** (see Figure D.22)

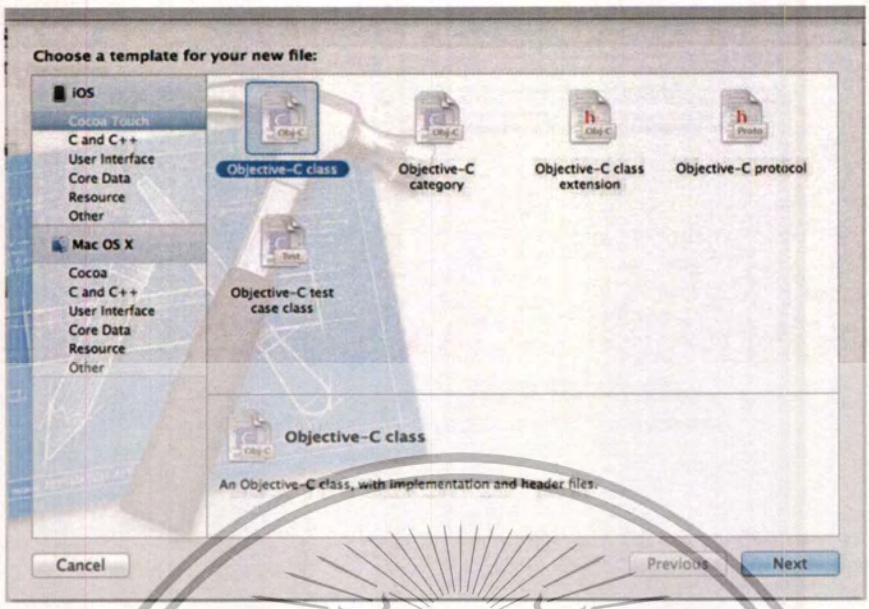


Figure D.22 Template for new file Window

24. Name this file "*Homepage*" then choose *UIViewController*. Click **Next button** (see Figure D.23)

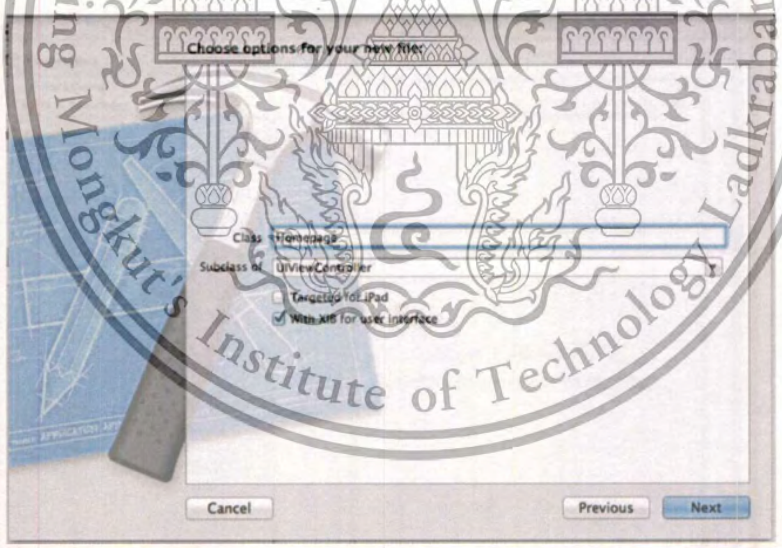


Figure D.23 Option for Homepage file Window

25. Choose the location that user need to save this file, then click **Create button** (see Figure D.8)

26. The *Homepage.XIB* will show on the screen (see Figure D.24)

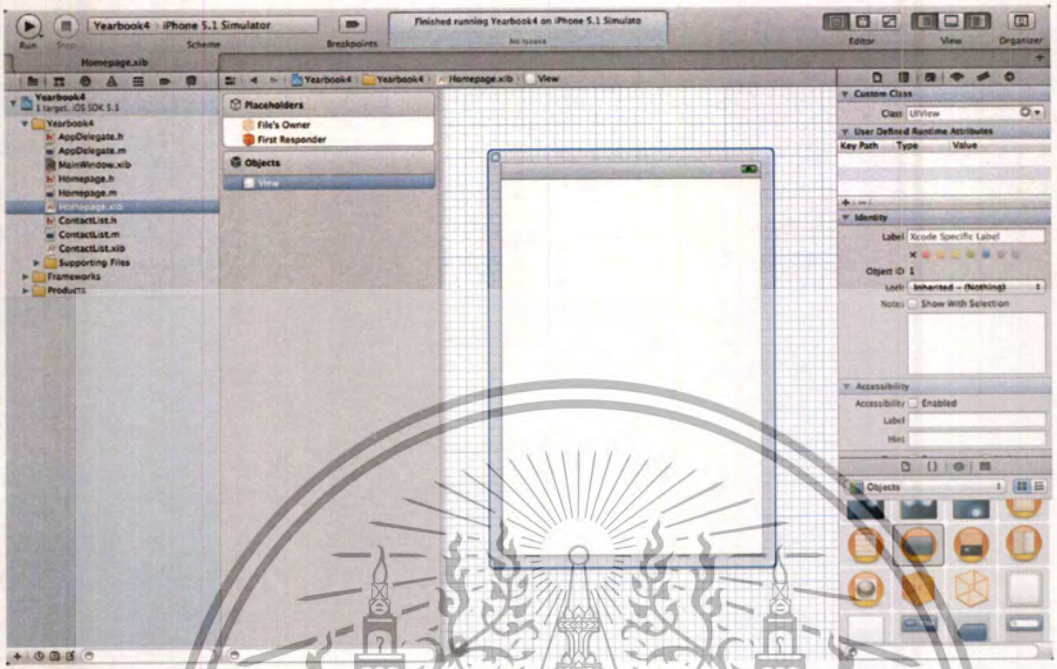


Figure D.24 *Homepage.XIB* interface builder window

27. Create new File, click on File ➔ New ➔ File

28. Click on *Cocoa Touch*, choose *Objective-C class*. Click **Next button** (see Figure D.22)

29. Name this file **"*ContactList*"** then choose *UITableViewController*. Click **Next button** (see Figure D.25)

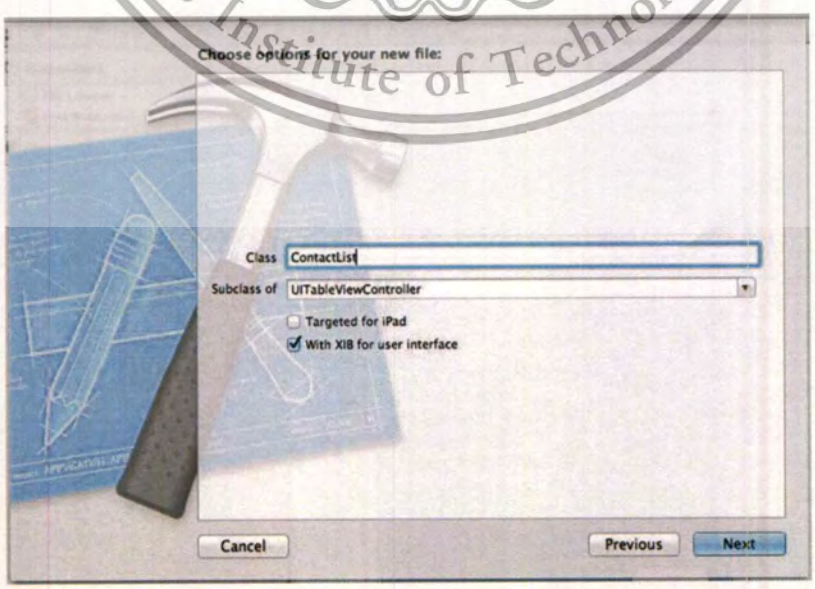


Figure D.25 Option for *ContactList* file window

30. Choose the location that user want to save this file, then click **Create button** (see Figure D.8)
31. The *ContactList.XIB* will show on the screen (see Figure D.26)
32. Go to *MainWindow*, Click on *Tab Bar Controller* (see Figure D.27-yellow section), then click on Utilities ➔ identity inspector ➔ custom class ➔ typing *Homepage* (see Figure D.27-red section)
33. Click on *Navigation Controller* ➔ *Tab Bar Controller* (see Figure D.28-yellow section), then click on Utilities ➔ identity inspector ➔ custom class ➔ typing *ContactList* (see Figure D.28-red section)

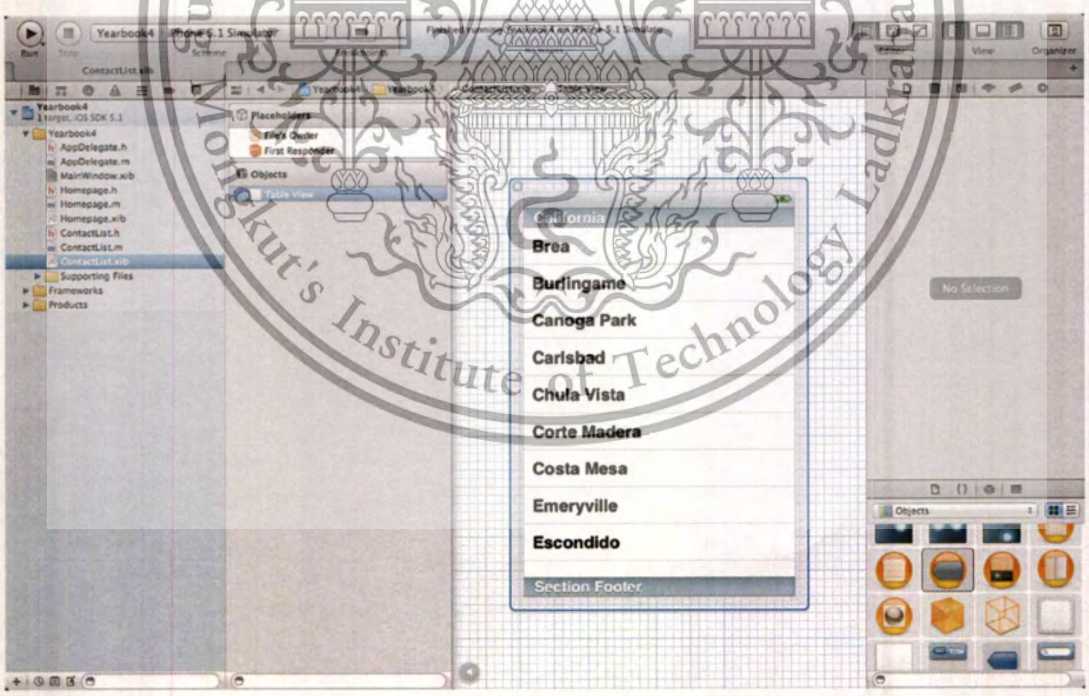


Figure D.26 *ContactList.XIB* **interface builder window**

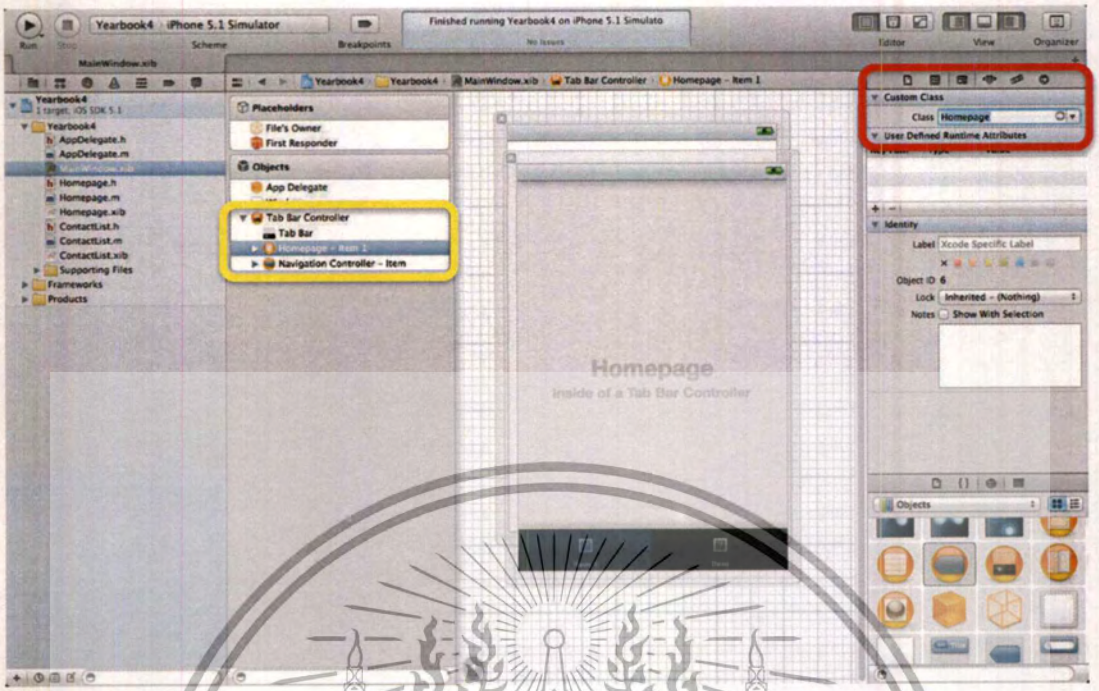


Figure D.27 *MainWindow.xib*

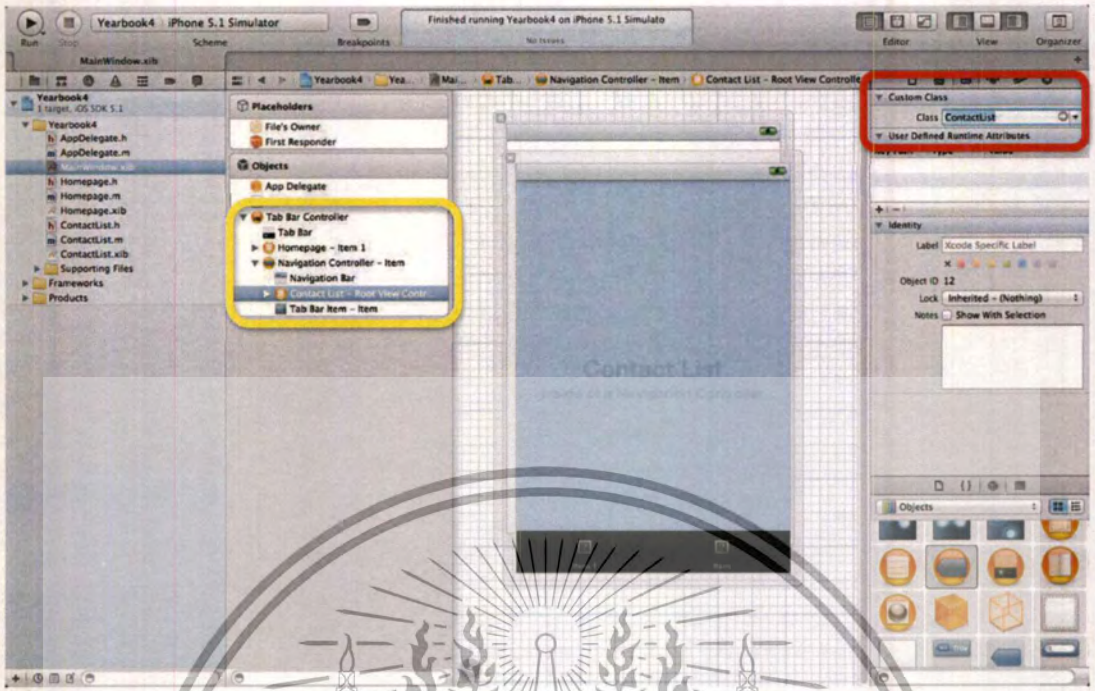


Figure D.28 *MainWindow.xib*

34. Go to home screen (see Figure D.4) then set the main interface as *MainWindow* (see Figure D.28-yellow section)

Figure D.29 Home Screen Window

35. Click on  button to build and run the application (see Figure D.29)

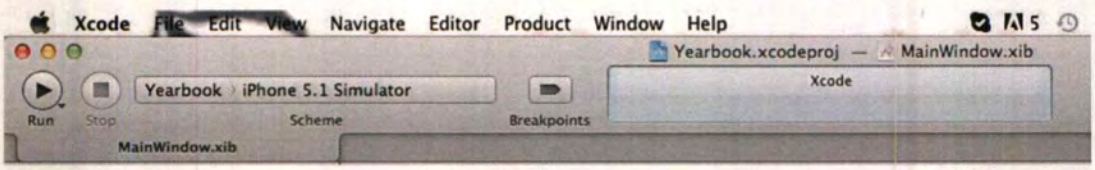


Figure D.30 Tool bar of Xcode program

36. After click on run button the iOS simulator will show on the screen (see Figure D.30)

Figure D.31 iOS simulator

37. Click on File ➔ Add File to "application name" ➔ choose file in the file directory ➔ click **Add button** (see Figure D.31)

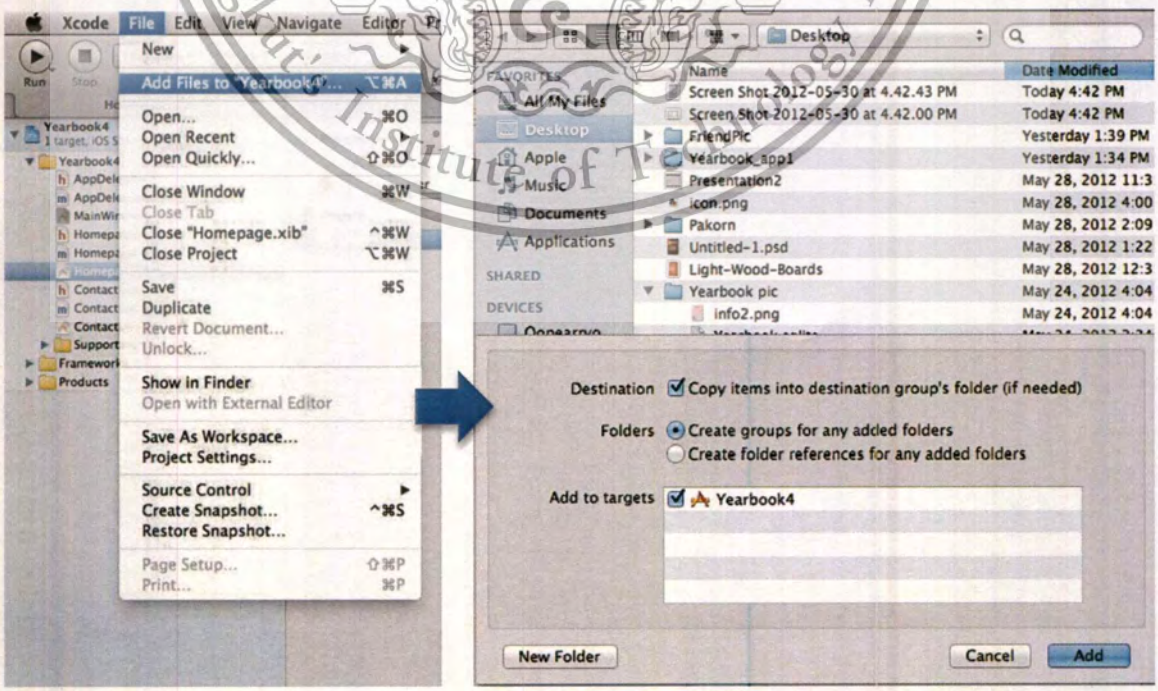


Figure D.32 Add file to Xcode

38. Go to **Homepage.xib** then create *UIImageView* to add the image for background (see Figure D.32) then drag *UIButton* to the screen for create button on Homepage (see Figure D.32-green section)

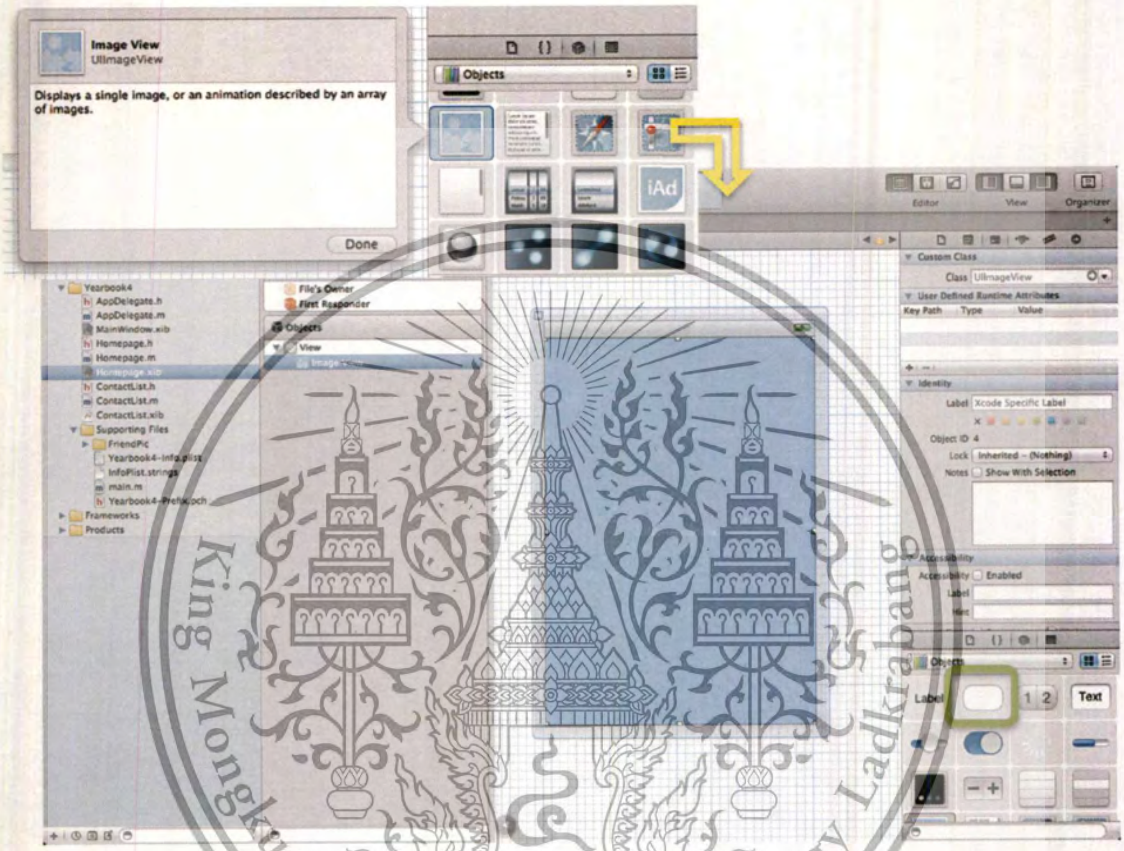


Figure D.33 UIImageview

39. Click on *Image View* (see Figure D.33-red section), then click on Utilities ➔ attribute inspector ➔ image ➔ typing *background Picture name.png* (see Figure D.33-yellow section).

40. Click on Button (see Figure D.34-red section), then click on Utilities ➔ attribute inspector ➔ image ➔ typing *button Picture name.png* or click on

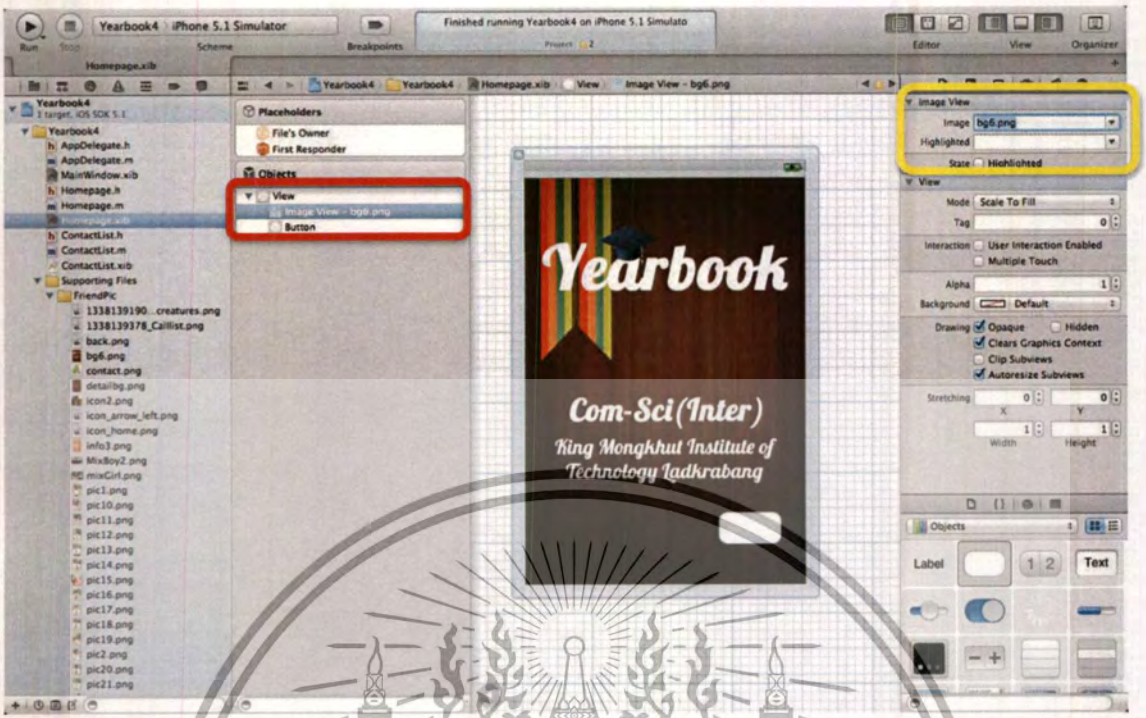


Figure D.34 Image view attribute inspector

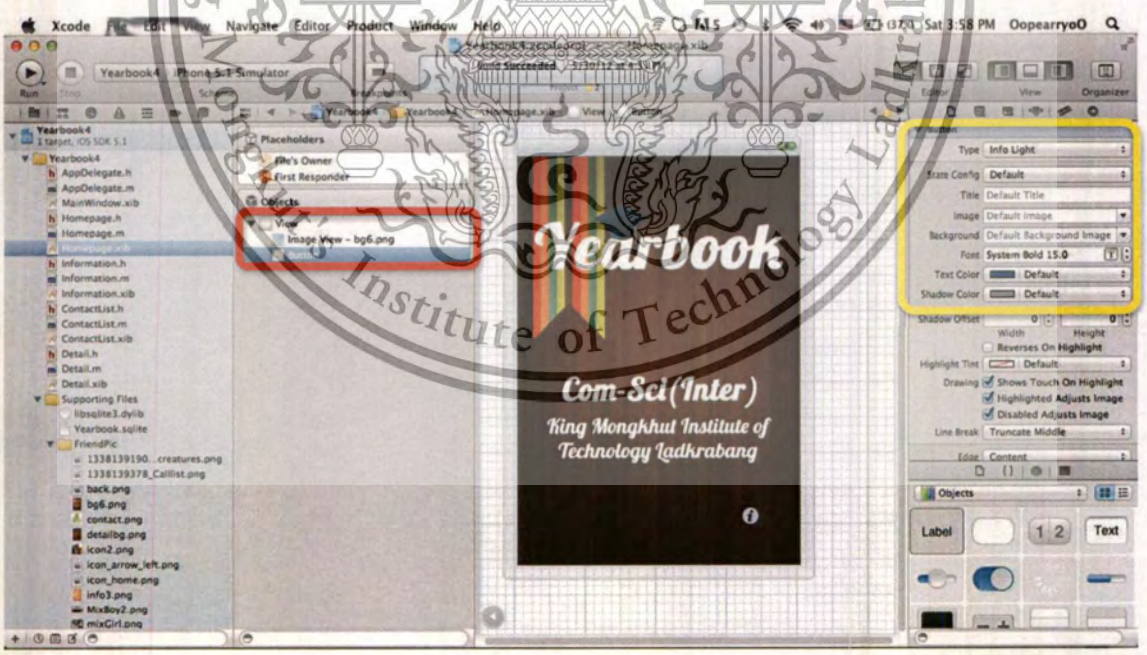


Figure D.35 UIButton attribute inspector

41. Click at run button, the first screen of the Yearbook project will appear as shown in Figure D.35.

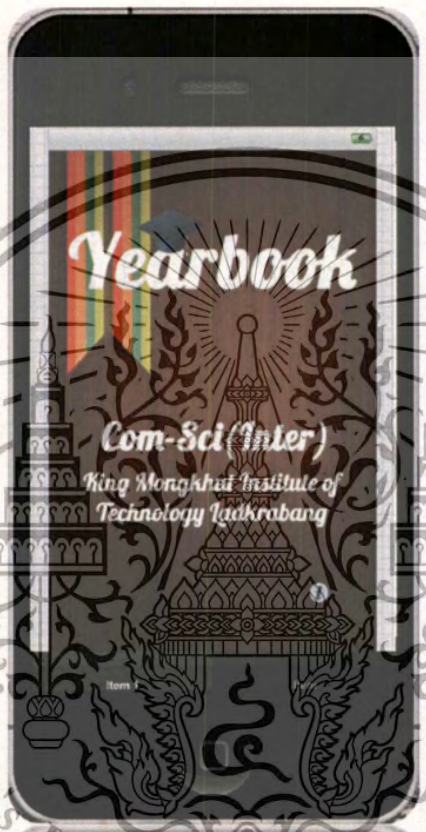


Figure D.36 Home screen of Yearbook Application

D.2 How to install the Application to iOS device

Our application uses the iOS Enterprise Program[1]. To install the application on the iOS device, the iOS developers needs 3 things: the developer certificate, the device ID and the app ID.

Developer certificate: An iOS application must be cryptographically signed by its developer before it can be run on an iOS device. Hence, developers have to obtain the certificate that authorizes them to sign first.

Device ID: The device ID is the ID of the apple device that developers want to install this application on it. Each apple device has an ID called the Unique Device Identifier (UDID).

App ID: An App ID is used to specify an application, or set of applications. Each iOS application must have the App ID.

The certificate, device ID and App ID are used to create a Provisioning Profile. This Provisioning Profile must be installed on each device that the developers run their application code *[2]*.

D.2.1 Request a Development Certificate

Steps to request for the certificate are as follows:

1. Choose Window ➔ Organizer.
2. Click Devices.
3. In the Library section, select Provisioning Profiles.
4. Click the Refresh button at the bottom of the window

Enter your user name and password and click Log in. After you log in to your account, a prompt appears, asking whether Xcode should request your development certificate.

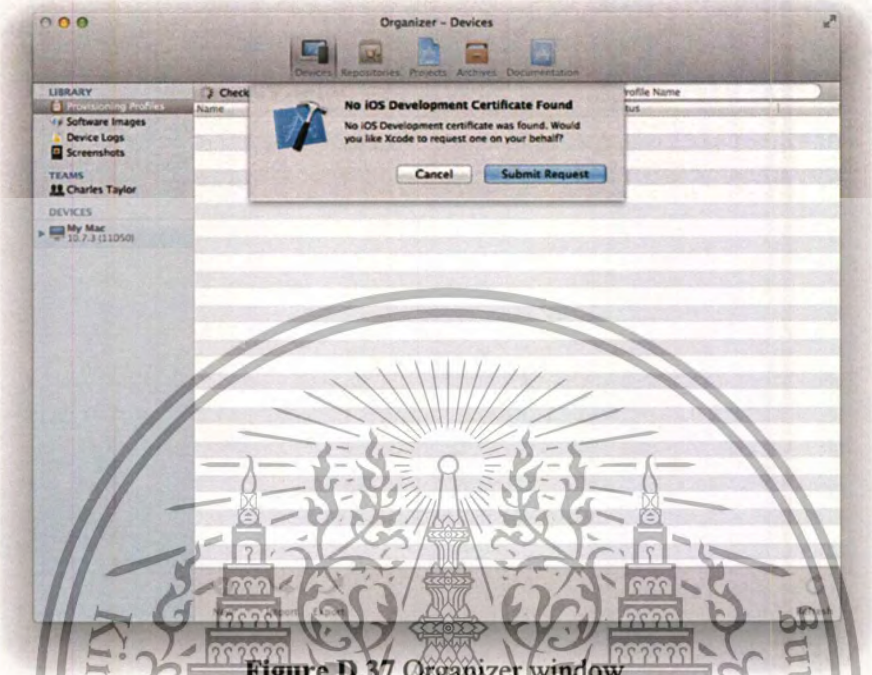


Figure D.37 Organizer window

5. Click the Submit Request button. The development certificate is added to your keychain and later added to the iOS Team Provisioning Profile. (More prompts may appear, asking whether Xcode should request other types of certificates. Click the Submit Request button for each prompt that appears.)

6. If a prompt appears, at the end of the refresh process, asking if you want to export your developer profile, click Export. The private keys for your certificates are stored in your keychain, and the public keys are stored in the portal. For this reason, you cannot refresh your provisioning profiles and certificates in Xcode to replace a missing private key in your keychain. Instead, you should back up your certificates after you create them and import them when you are missing a private key or move to another Mac.

Enter a filename and password, and click Save. Because the file contains your digital identity which can be used to sign apps in your name, it is encrypted and password protected.

D.2.2 **Provision Your Device**

Adding the device to the Provisioning Profile can be done according to the following steps:

1. Connect your device to your Mac.
2. Open the Devices organizer (Window → Organizer → Devices).
3. In the Devices section, select your iOS device.
4. Click the "Use for Development" button. The first time you add a device ID to your account, if the device was used for development in the past, the "Use for Development" button may not appear. If this happens, click "Add to Portal" at the bottom of the screen instead.

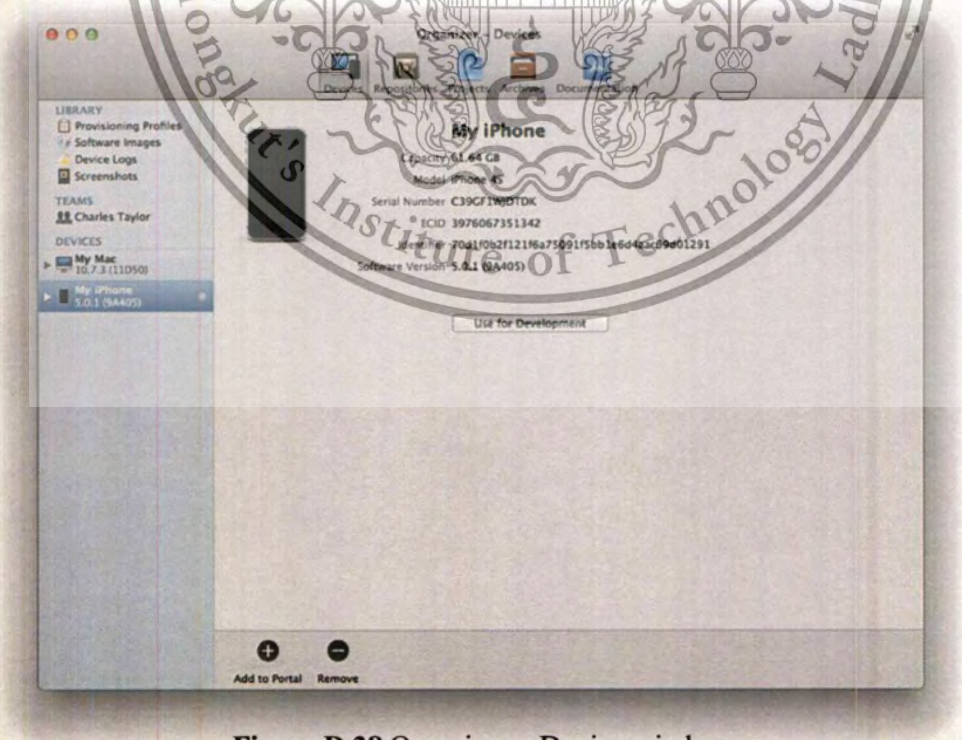


Figure D.38 Organizer – Device window

D.2.3 Code Sign Your App

Code signing is a process for the developers to sign their iPhone applications to certify that the applications are released by them [3]. Steps to sign the applications are as follows:

1. Select the project.
2. Click Build Settings
3. Click All.
4. Type Code Signing in the search field in the Build Settings pane of the project editor.
5. From the Code Signing Identity pop-up menu, in the iOS Team Provisioning Profile section, choose the certificate that begins with "iPhone Developer:" followed by the developer name.

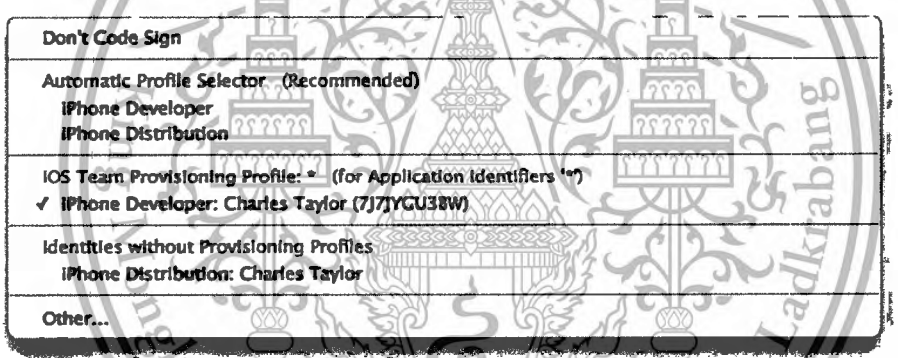


Figure D.39 Code sign menu list

D.2.4 Launch Your App on the iOS Device

To launch the application on the device, follow the steps below.

1. Choose Product > Edit Scheme to open the scheme editor.
2. Select your device from the Destination pop-up menu. When you connect an iOS device with a valid provisioning profile into your Mac, its name appears as an option in the destination Scheme pop-up menu.

D.3 Reference

[1][Online] https://developer.apple.com/programs/ios/enterprise/

[2][Online] http://developer.apple.com/library/ios/#documentation/ToolsLanguages /Conceptual/DevPortalGuide/Introduction/Introduction.html#// apple_ref/doc/uid/TP40011159-CH1-SW1

[3][Online] http://developer.apple.com/library/mac/#documentation/ToolsLanguages /Conceptual/OSXWorkflowGuide/CodeSigning/CodeSigning.html