

City Guide over Android

TDT4520 Specialization Project

Hanjie Shu

Spring, 2010

Supervisor: John Krogstie, Jacqueline Floch



**NTNU
Norwegian University of
Science and Technology**

Department of Computer and Information Science, IDI

Abstract

The goal of the project is to explore how to realize a mobile city guide using the Android platform, including a prototype of the city guide.

The project uses the research method Design Science. Through designing and implementing an artifact (i.e. prototype of city guide), the goal of the project is reached. Finally, the project is evaluated in four aspects including platform evaluation, general functional evaluation, scenario evaluation, and non-functional evaluation.

The prototype implemented includes basic functionalities of city guide such as showing a map, locating points of interest (POIs) on a map, locating location of a user, retrieving information of POIs, add reviews about POIs, plan a tour, support communication (e.g. phone, short message), show route direction to POIs, add reminder, and choose different kinds of POIs to show on map.

Moreover, the project has explored how to integrate current technologies like Google Calendar, Google Map, Browser, Contact application and Phone application into the prototype.

As well, the project has investigated non-functional aspects including extendibility, tailorability, and usability.

Overall, the project presents a comprehensive understanding of how to realize city guide on the new mobile platform Android.

Preface

This report documents the project work performed in the subject TDT4520 “Program and Information Systems, Specialization Project” by Hanjie Shu. The project counts for 15 credit points and is executed in the first semester of the last year of Master Program of Information Systems in The Norwegian University of Technology and Science, NTNU.

The project was defined through the UbiCompForAll research project, coordinated by SINTEF in Trondheim. UbiCompForAll started in October 2008, and the name is short for “Ubiquitous Service Composition for All Users”. And the project was defined by Jacqueline Floch, project manager of the UbiCompForAll project. The project intends to explore the realization of a City Guide on the Android platform.

I would like to thank my main advisor at SINTEF, Jacqueline Floch, and my supervisor at NTNU, John Krogstie, for their support and feedback.

Trondheim, 15th June 2010

Hanjie Shu

Table of Contents

Chapter 1: Introduction	1
1.1 Project Context.....	1
1.2 Problem Definition.....	1
1.3 Project Contribution.....	2
1.4 Report Outline.....	2
Chapter 2: Background	3
2.1 Existing Mobile City Guides and Similar Solutions.....	3
2.1.1 Information Guide	4
2.1.2 Events Guide.....	5
2.1.3 Navigation Software.....	7
2.1.4 Tailorable Software.....	8
2.1.5 Others.....	9
2.2 Mobile Application Development Technologies	10
2.2.1 WAP	10
2.2.2 J2ME	10
2.3 Android	12
2.3.1 Highlights of Android	12
2.3.2 Introduction of Android	13
2.3.3 Application Fundamentals	15
2.4 Location sensing technologies	18
Chapter 3: Research Method	19
3.1 Research Method.....	19
Chapter 4: Requirement Specification.....	22
4.1 Scenarios	22
4.1.1 Scenario 1: Corporate Group.....	22
4.1.2 Scenario 2: Family.....	23
4.2 Requirement Specification	24
4.2.1 Functional Requirements	24
4.2.2 Non-Functional Requirements	26
Chapter 5: Solution.....	27
5.1 General Architecture.....	27
5.2 Implementation of Functions	29
5.2.1 Component Diagram.....	29
5.2.2 Map and POIs and Review.....	31
5.2.3 Get My Location.....	32
5.2.4 Reminder.....	33
5.2.5 Search.....	35
5.2.6 My Tour plan	36
5.2.7 Driving Direction.....	37
5.2.8 Communication	38

5.3 Class Diagram	39
5.4 Data Model.....	41
Chapter 6: Evaluation	43
6.1 Platform Evaluation.....	43
6.2 General Functional Evaluation.....	43
6.3 Scenario Evaluation	45
6.4 Non-Functional Evaluation	57
6.4.1 Flexibility	57
6.4.2 User Interface Friendly.....	57
6.4.3 Combination of Existing Innovative Technologies	57
Chapter 7: Conclusion and Further work	58
7.1 Achievements	58
7.2 Further Work	59
APPENDIX	60
A: References.....	60
B: Code Examples	62
C: POIs and Events XML Document Example	64
D: UI Design in XML document	67
E: Relevant Source Code.....	68

Figures

Figure 2-1: Demo of Vindigo [Vindigo 2010].....	4
Figure 2-2: The architecture of solution of [Hansen 2007];.....	5
Figure 2-3: Demonstration of [Blakstad 2008] on Nokia N95.....	6
Figure 2-4: Screenshot of Wayfinder Navigator.....	7
Figure 2-5: myMytileneCity Guide [Kenteris 2007].....	8
Figure 2-6: Architecture of Android platform [Android 2010].....	13
Figure 2-7: The Lifetime of Activity	16
Figure 2-8: Location-sensing technologies [Hazas 2004].....	18
Figure 2-9: J2ME Architecture [Helal 2002].....	11
Figure 5-1: General Architecture of Solution	27
Figure 5-2: Component Diagram	29
Figure 5-3: Data Model.....	41
Figure 5-4: Message Sequence Chart of Map and POIs and Review	31
Figure 5-5: Message Sequence Chart of Automatic Localization.....	32
Figure 5-6: Message Sequence Chart of Reminder.....	33
Figure 5-7: Reminder Implementation.....	34
Figure 5-8: Message Sequence Chart of Search.....	35
Figure 5-9: Message Sequence Chart of My Tour Plan.....	36
Figure 5-10: Message Sequence Chart of Driving Direction	37
Figure 5-11: Class Diagram of Solution	39
Figure 6-1: Start Application	45
Figure 6-2: Contact.....	47
Figure 6-3: Menu	46
Figure 6-4: Search.....	49
Figure 6-5: Result of Search about Church.....	49
Figure 6-6: Tap on bubble of POI	50
Figure 6-7: Description of POI about Nidaros Cathedral.....	50
Figure 6-8: Review of POI.....	50
Figure 6-9: Add Review.....	50
Figure 6-10: My Tour Plan	51
Figure 6-11: Reminder.....	51
Figure 6-12: Add new Reminder.....	52
Figure 6-13: Add title of Reminder.....	52
Figure 6-14: Add Location of Reminder.....	52
Figure 6-15: Set time of Reminder	52
Figure 6-16: Set Date of Reminder	53
Figure 6-17: Show Reminder Result.....	53
Figure 6-18: Get My Location	54
Figure 6-19: The result after choosing one Sight in Tour plan	54

Figure 6-20: Show Direction to one Sight.....	55
Figure 6-21: Contact with somebody	56
Figure 6-22: New Contact.....	47
Figure 6-23: Group.....	48
Figure 6-24: new group.....	48
Figure 6-25: Add contact to group.	48
Figure 6-26: Unsuccessful to get Location.....	54
Figure 6-27: Firstly go to get current position.....	55

Tables

Table 3-1: Guideline for Design Science Research [Hevner 2004]	19
Table 4-1: Map in Functional Requirements.....	24
Table 4-2: Information Retrieval in Functional Requirements	25
Table 4-3: Communication in Functional Requirements	25
Table 4-4: Review in Functional Requirements.....	25
Table 4-5: Reminder in Functional Requirements	25
Table 4-6: Plan Tour in Functional Requirements	25
Table 4-7: Non-Functional Requirements.....	26
Table 6-1: Map Evaluation	43
Table 6-2: Information Retrieval Evaluation	44
Table 6-3: Communication Evaluation	44
Table 6-4: Review Evaluation.....	44
Table 6-5: Reminder Evaluation	44
Table 6-6: Plan Tour Evaluation	44

Chapter 1: Introduction

The chapter presents project context, project goal, project contribution and report outline.

1.1 Project Context

The project is given in the context of UbiCompForAll [UbiCompForAll 2010], which is a project funded by the Research Council of Norway; it started in October 2008 and is meant to last 4 years. It contains several partners SINTEF ICT, NTNU, Wireless Trondheim AS and others. The goal of UbiCompForAll is to provide a solution to enable end users (non-IT professional) compose services according to their personal needs in ubiquitous computing environment.

Today, people want to adapt their living environment to intelligent environment, which includes a powerful infrastructure and some intelligent objects. People perform their tasks through the intelligent environment. However, it leads to an issue about how users customize computing activity for themselves, because sometimes there are a great number of services around them, how to manage the complexity and tailor them to what users really want. End users are considered non-IT professional. Then, developers should provide composition tools, which enable users to compose services by themselves. It is just the goal of UbiCompForAll. The challenge is that service composition framework must be sophisticated enough to support correct service composition, as well, must be intuitive enough for ordinary end-users.

The platform of Android [Android 2010] has been chosen to implement mobile services in UbiCompForAll. The project is intended to explore the realization of city guide over Android. The result of the project can be used as the start point for service composition research, which meets the goal of UbiCompForAll.

1.2 Problem Definition

The project is to investigate the realization of tailorable City Guide over the platform of Android. One City Guide should contains some functionalities like automatic localization, navigation support, retrieving information from points of interests, setting reminder, adding reviews, communication support and so on. Moreover, the project should explore current innovational technologies as many as possible and put them as building blocks like Google Map, Google Calendar, and Twitter. And the project also needs to see how they can be tailored and composed together. Finally, the project will result in the demonstration of prototype of city guide.

1.3 Project Contribution

The main contribution of the project is to present a comprehensive understanding of how to realize city guide on the new mobile platform Android. Now we can see many realizations of city guide on other platforms such as Windows Mobile, iPhone, J2ME and so on. However, there is few city guides over Android. The final prototype covers basic functionalities of city guide such as showing map, locating points of interest (POIs) on map, locating user's location, retrieving information of POIs, add review about POIs, plan a tour, communication, show route direction to POIs, add reminder, and choose different kinds of POIs to show on map.

1.4 Report Outline

Chapter 2: Background describes existing mobile city guides and similar solutions, and presents mobile application development technologies like WAP, J2ME, and gives concrete introduction of Android, finally introduces wireless Internet access technology and location sensing technology.

Chapter 3: Research Method introduces the research method Design Science and describes relevant points between Design Science and the project.

Chapter 4: Requirement Specification presents scenarios used in the project, and functional requirements and non-functional requirements.

Chapter 5: Solution gives the solution architecture of the project, and solution of each function.

Chapter 6: Evaluation presents platform evaluation, general functional evaluation and scenario evaluation.

Chapter 7: Conclusion and Further work presents the achievement of the project, and the points needed to be improved.

Chapter 2: Background

The chapter describes existing mobile city guides and similar solutions, and presents mobile application development technologies like WAP, J2ME, and gives concrete introduction of Android, and comparison between these different mobile platforms, finally introduces wireless Internet access technology and location sensing technology.

2.1 Existing Mobile City Guides and Similar Solutions

In this chapter, I shortly describe some city guides or related applications, some of them are products in the market, and some are for research or students' projects. We can summarize solution methods and functional requirements from them. According to their features of functionalities implemented, I divide them into different groups:

Information Guide is focus on providing information like restaurants, bars, museums, movies, ATM, parking lots and so on in a city. Usually, they don't have automatic localization.

Event Guide emphasizes on providing information about current events around user's position in a city. Sometimes, they also provide information about nearby bars, hotels, parking lot and so on.

Navigation Software puts navigation support as its main feature.

Tailorable Software enables users to tailor their visit based on their personal interest, and tailors the information they want.

In addition, I list other applications briefly, some of them can let users to add review, some support audio navigation, some are the solutions based on browser.

2.1.1 Information Guide

Vindigo

Vindigo [Vindigo 2010] is software for PDA (Palm and Pocket PC/Windows Mobile). The software helps users know about great restaurants, bars, movies, and museums in major U.S. cities and find ATM and parking lots nearby. It also tells users how to get where they want to arrive with the color map. But users have to input locations by themselves, the product lacks automatic localization. And users can submit reviews of restaurants, bars and so on. The content in the software can be updated when the PDA is connected to computers, which is connected to Internet. Generally, it is not as advanced as some current popular products. However, it provides information, which is very useful for users. It is very helpful for me to design functions in City Guide for how to meet users' need. There is a picture attached below of the product.



Figure 2-1: Demo of Vindigo [Vindigo 2010]

2.1.2 Events Guide

Cultural Guide

[Hansen 2007] is a project in NTNU with the aim of creating a mobile application for getting current or following happening event in Trondheim. Users can find some events showed on a map on their mobile phones with the application, these events will happen in the following hours near where they are. The functionalities implemented in the project could be a part of functional requirements of my project. The architecture of the project is showed in the Figure 2-2 below. Cell phone is positioned with GPS transmitter installed in it; it sends Http Request (1) in Java (Development platform chosen for the project is J2ME) to PHP server, which retrieves required data (2) from Utguiden Database. Utguiden [UtGuiden 2010] is a cultural guide managed by the local newspaper in Trondheim, Adresseavisa. It contains all forms of events hosted in Trondheim and is accessed through the website of Adresseavisa. The project uses an existing framework of developing map application for mobile device. The framework can display a detailed map of Trondheim; and it is possible to register information at specific locations.

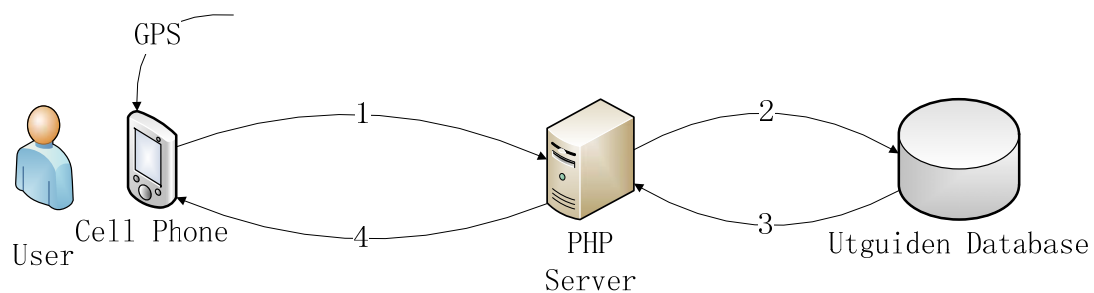


Figure 2-2: The architecture of solution of [Hansen 2007];

Mobile City Guide

Blakstad [Blakstad 2008] created a mobile city guide of Trondheim, which shows user's current position on map, shows current events or places to eat or drink near the current position. Users can read further information about them. Users also can navigate around the map. Figure 2-3 below shows the software running on Nokia N95. Smiley face represents user's current position, the number 1, 2, 3 stands for points of interest near the user. The city guide uses maps and POI-database from IPos [Ibrahim 2008], a free map based on the OpenStreetMap project and a map from Norge Digital. The source of information used in the project is from UtGuiden [UtGuiden 2010] that is also used in [Hansen 2007] as mentioned in last section for providing current events, and Vibb [Vibb 2010] as information source of business. It can use GPS data for localization, and Wi-Fi localization from the solution of GeoPos [GeoPos 2010] that is a project and company initiated at NTNU.



Figure 2-3: Demonstration of [Blakstad 2008] on Nokia N95

2.1.3 Navigation Software

Standalone GPS device is popular today, especially for providing navigation when driving, because it is easy to use without too many functions and more and more people drive cars. It is so useful to direct people to arrive in their destinations. It includes some information like driving routes, gas stations, points of interest. Meanwhile, there are many mobile phone with GPS installed now, like Nokia 6110 Navigator [Nokia 6110]. Their manufacturers also provide navigation application with such information as described above. Some third party software companies also produce navigation software for mobile phones.

Wayfinder Navigator

Wayfinder is a leading supplier of location and navigation service for mobile phones. Wayfinder Navigator [Wayfinder 2010] is its flagship product through some years and transforms mobile phones to more useful than the ordinary GPS. In addition to the functions of Vindigo [Vindigo 2010], it provides information about the real traffic information, and even gives alternative route in the case of traffic jams. And users get warned when there are speed cameras. Users also get voice instruction and weather reports. The software helps users share their favorites with their friends or family via SMS or Email. Finally, it can update information and latest map through mobile internet. Regrettably, Wayfinder has stopped development and maintenance of the product due to be acquired by Vodafone. Its relevant navigation product has been installed into Vodafone 360 handsets. The Figure 2-4 below is a screen shot from Wayfinder Navigator. Overall, Wayfinder Navigator is powerful and includes many kinds of functions. It is a good model for my project.



Figure 2-4: Screenshot of Wayfinder Navigator

Ovi Map

Ovi map [Ovi map 2010] is the navigation application running on many Nokia phones. It covers map of over 70 countries all over the world. It supports driving navigation, walk navigation, map update, event guides, and Lonely Planet guide [Lonely Planet 2010], which is the world largest provider of electronic tourist guides.

2.1.4 Tailorable Software

myMytileneCity Guide

There is a J2ME-based mobile tourist guide application myMytileneCity Guide presented in [Kenteris 2007]. Firstly the guide lets users input information that they are interested in. Then the information is sent to remote server. The remote server generalized JAR file including the information. Finally, it sends back to users, and users installed it on their mobile devices. The architecture of the tourist guide is showed in the Figure 2-5 below. Server notifies end users update information through SMS message. If end users agree, their mobile devices connect to Server to download new content. The mechanism of dynamic installing according to the interest of end users meets the feature of tailorability well.

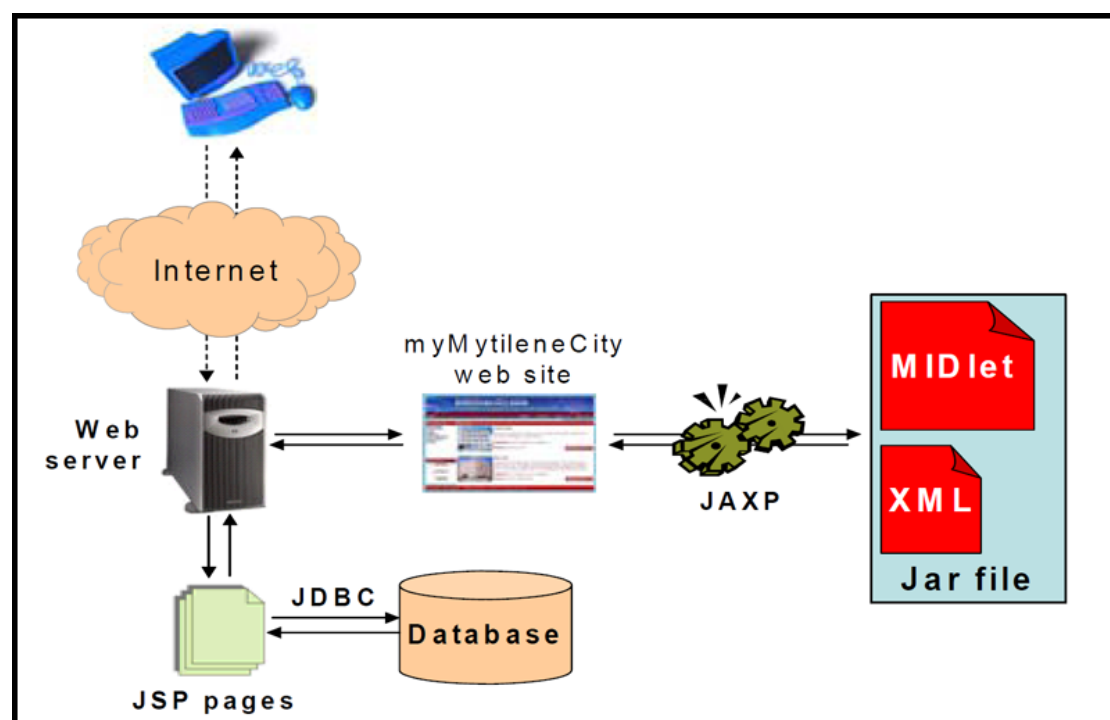


Figure 2-5: myMytileneCity Guide [Kenteris 2007]

GUIDE

GUIDE [Cheverst 2000] is a tourist guide, which has the important feature of being tailorable. Users can tailor their visiting route based on their interests. In addition, the guide can tailor information, provide special information to end users according to their personal preference (e.g. somebody is interested in museum or architecture), environmental context, and the time of the day. And the guide can notify users the dynamic information, for example, when one tourist point is closed earlier than planned due to emergency.

2.1.5 Others

AudioTravel [AudioTravel 2010] has mobile travel guide including some basic functions as described above, especially, it provides audio player to guide users in the map.

CityGuide [AOL 2010] enables users see categories of venue across US such as restaurants, bars and clubs, all voted “Best” by visitors.

Traveldodo [Traveldodo 2010] provides a free mobile city guide, which covers over 500 cities in the world. It is compatible with almost all the mobile phones after 2006. It is an off-line application. It allows users to add reviews.

Schmap [Schmap 2010] enables iPhone, iPod Touch, Nokia users to visit the website www.schmap.com with the browser in their mobile devices to get city guide with dynamic map, pictures, review, and local search.

2.2 Mobile Application Development Technologies

There are three major mobile application development technologies presented in [Read 2003]: Wireless Application Protocol (WAP) and NTT DoCoMo's i-mode, and Sun's Java 2 Micro Edition (J2ME). The second technology i-mode is a kind of wireless Internet service with strong business model, different with WAP that is a protocol.

2.2.1 WAP

WAP (www.wapforum.org) is a free, open specification that lets wireless devices easily interact with service and each other. WAP 2.0 adopts existing web standards and lets developers create applications that feature animation, streaming media, and music download. In WAP 2.0, developers write content in XHTML using the XHTML basic profile. WAP 2.0 is backward compatible with the previous standard, WAP 1.x, which uses Wireless Markup Language (WML) rather than XHTML for document formatting.

2.2.2 J2ME

J2ME is a popular mobile development platform now. J2ME is from Sun Microsystems and targets on consumer electronics, portables and embedded devices. The architecture of J2ME consists of three layers: Java Virtual Machine, Configurations, and Profiles. There is the Figure 2-9 from [Helal 2002] describing the architecture of J2ME. We can see there are two columns in the diagram. They represent two sub architectures of J2ME respectively. Left is considered for devices that are always connected but relatively resource poor such as set-top boxes (e.g. satellite TV receiver). Right is for the devices such as mobile phones. The kilobyte Virtual Machine at the bottom of right is smaller runtime environment for resource-constrained devices. It is in the range of 40-80 Kbytes. On KVM, Connected and Limited Device Configuration (CLDC) defines a standard Java platform for small, resource-constrained, connected device and enables the dynamic delivery of Java application and content to those devices. The top one is MIDP, which is set of Java APIs related to interface, persistence storage, networking, and application model.

J2ME application starts in MIDlet, which contains the constructor method as well as the methods startApp(), pauseApp(), and destroyApp(). It calls the first method when

application starts or restarts. It calls the `pauseApp()` when the phone's idle or paused mode and calls `destroyApp()` right before it is unloaded.

J2ME is best used for complicated or interactive applications, or when WAP and i-mode is not available and not suitable.

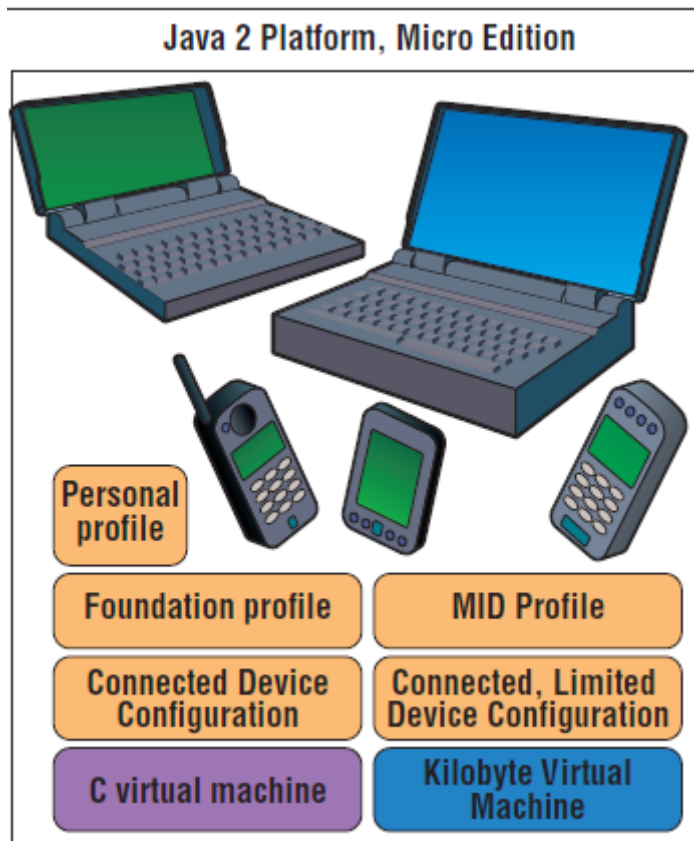


Figure 2-9: J2ME Architecture [Helal 2002]

2.3 Android

Android is Mobile Platform developed by Google. Developers create applications in Java on the platform. It includes some important features like 3D graphics, Media support for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF), GSM Telephony, Bluetooth, 3G, Wi-Fi, GPS depending on hardware capability of mobile devices.

2.3.1 Highlights of Android

Firstly, Android is open-source platform. Secondly, Android enables reuse of components. For example, there is one component for editing text files in one application; other applications can make use of the component as needed, of course, the application has released the permission that other applications can use it. In J2ME [Sun 2010], one application implements most functionalities in a single midlet extending from *Midlet* base class but Android contains different components types of Activity, Service, Content Provider, Broadcast Receiver in one application and they can communicate through sending intent message, J2ME is not easy as Android to write modular code. Thirdly, Android can be native access to Google map infrastructure. And it supports GPS localization. Fourthly, its network can work in the way of Bluetooth, 3G or Wi-Fi. Finally, it provides Widgets classes and Layout classes for designing UI. Developers can write UI in XML layout file as described in APPENDIX D.

2.3.2 Introduction of Android



Figure 2-6: Architecture of Android platform [Android 2010]

There is a figure (Figure 2-6) of the architecture of Android platform showed above.

For the layer Applications at the top, Android ship with some core applications including Email, SMS, Calendar, Map, Browser, Contacts and others. Applications are developed in Java.

Different from other mobile platforms like J2ME, Application Framework below Applications enables reuse or replacement of components. For example, there is the component of displaying image in one application; other applications can make use of the component if needed. Of course, there is permit protocol between them.

Libraries include a set of C/C++ libraries used by various components of Android system. These are exposed to developers through the application framework. The

core libraries includes browser engine library, 2D and 3D graphics libraries, media libraries, SQLite that is a powerful and lightweight relational database engine available to all applications and so on. For Android Runtime, [Android 2010] says “Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.” Each Android application runs on its own process, with its own instance of Dalvik Virtual Machine. [Android 2010] says “The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.”

For Linux Kernel, [Android 2010] says “Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.”

2.3.3 Application Fundamentals

The part presents main concepts for application development. There are four types of application components: Activity, Service, Broadcast Receivers, and Content Provider. Android application may consist of one or several of these components types.

Activity presents user interfaces that users will interact with. For example, in text messaging application, one activity presents the user interface to let users write message to others. All activities are written through extending *Activity* base class.

Service does not have user interface and run in background, for example, play background music. Service extends *Service* base class.

The component **Broadcast Receivers** receives and reacts to broadcast announcements. For example, when low battery is low, the information is needed to inform users. All the receivers extend *BroadcastReceiver* base class.

The final one, **Content Provider** stores data and provide data. The content provider extends *ContentProvider* base class but applications do not call directly to the methods in *ContentProvider*, instead they call methods in the object *ContentResolver*, which call *ContentProvider*. Android provides the following four mechanisms for storing and retrieving data: *Preferences*, *Files*, *Databases*, and *Network*.

The Android API contains support for creating and using SQLite databases. Each database is private to the application that creates it. The *SQLiteDatabase* object represents a database and has methods for interacting with it — making queries and managing the data. To create the database, call *SQLiteDatabase.create()* and also subclass *SQLiteOpenHelper*. All databases, SQLite and others are stored on the device in */data/data/package_name/databases*.

Except that the Content provider is activated by a request from *ContentResolver*, other three components are activated by asynchronous messages called *intents*. There is code example of triggering intent in APPENDIX B.

Android must know that application components exist before it can start the application components. So, the components written in applications should be registered in the *manifest* file, which is bundled into Android package, and is XML structured, and named *AndroidManifest.xml* for all applications.

Next, I want to say some about the lifecycle of components; it is important to understand lifecycle before developing applications. The Activity lifecycle is showed below in Figure 2-7. The colorful ovals are major states of Activity. The

rectangles are possible operations in the process of transitions between different states. There is a code example in APPENDIX B. The entire lifetime of one activity starts at the `onCreate()` and finishes on `onDestroy()`, where release all the remaining resources. If one activity is between `onResume()` and `onPause()`, it is foreground and interact with users, at the most top of all the activities. When one activity is between `onStart()` and `onStop()`, it is visible but may not be in the foreground and interact with users. During the time, the activity can maintain resources.

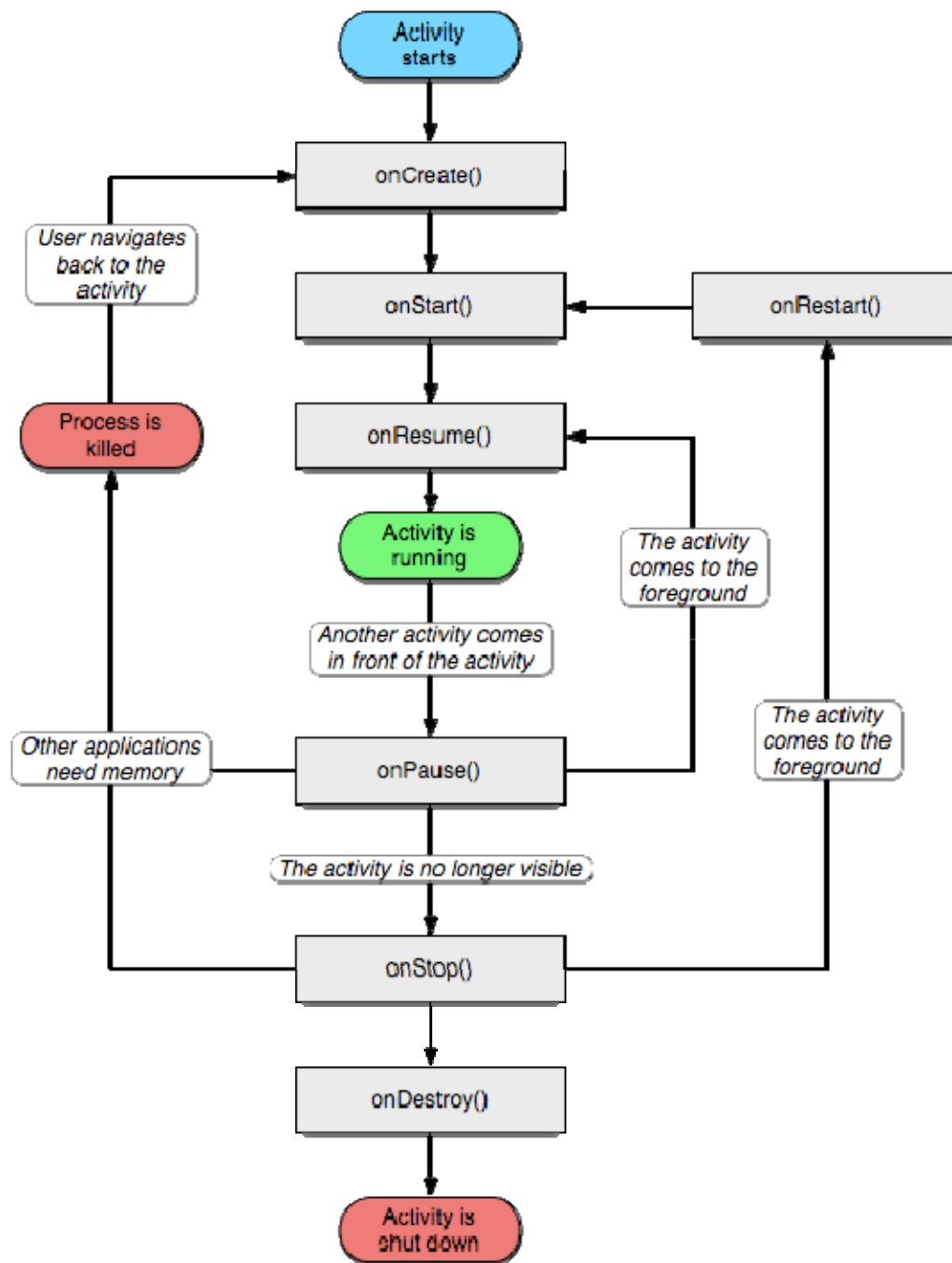


Figure 2-7: The Lifetime of Activity

The lifetime of Service is divided into two kinds of ways. One is that it is started standalone. The other one is that it is operated programmatically using an interface.

Broadcast Receiver is considered to be active while executing the method *OnReceive()* in the component, otherwise it is inactive.

Finally, UI can be designed in XML layout document including layout and widgets, there is an example showing in APPENDIX D.

2.4 Location sensing technologies

GPS (Global Positioning System) is the most widely known location-sensing technology today. A GPS receiver estimates position by measuring satellite signal's time difference of arrival. The US Department of Defense maintains the expensive satellite infrastructure in earth orbit. It is said in [Patterson 2003] that there are several reasons why GPS is not a universally applicable location sensing mechanism. Firstly, it does not work indoor, particularly in steel-framed building. Secondly, GPS use an absolute coordinate system, whereas some applications (for example, guidance systems for robotic equipment) require coordinate relative to specific objects. Finally, the specific component needed for GPS impose weight, cost and energy consumption requirements that are problematic for mobile hardware. In addition, GPS's performance degrades in high-rise urban areas, and receivers have a relatively long start-up time.

As a consequence, other location sensing technologies is developed. Wi-Fi Localization is one of them. It uses algorithms to compute localization based on data from Wi-Fi access points. In [Hazas 2004], there is the Figure 2-8 showing comparison of several location sensing technologies including GPS and Wi-Fi and mobile phones. We can see from the figure that usually GPS has higher accuracy than Wi-Fi and mobile phones. Actually, Wi-Fi accuracy depends on how dense the access points are positioned. However, Wi-Fi can work indoor in a way that GPS cannot do. And Wi-Fi also can cover across the whole large metropolitan area.

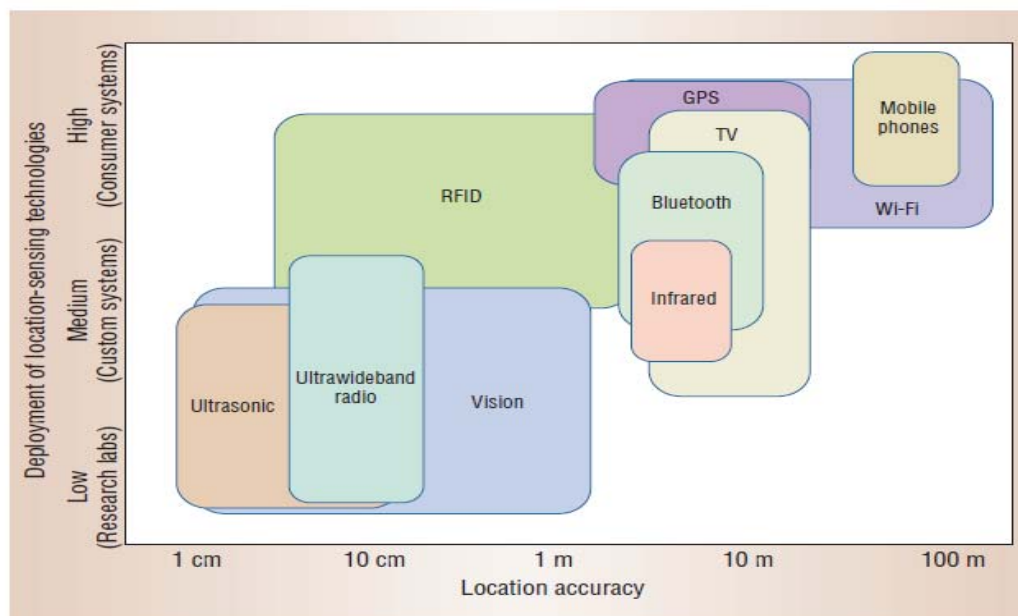


Figure 1. Location-sensing technologies. Each box's horizontal span shows the range of accuracies the technology covers; the bottom boundary represents current deployment, while the top boundary shows predicted deployment over the next several years.

Figure 2-8: Location-sensing technologies [Hazas 2004]

Chapter 3: Research Method

3.1 Research Method

The research method used in the project is Design Science. The fundamental principle of design science research is that knowledge and understanding of a design problem and its solution is acquired in the building and application of an artifact. There is the table from [Hevner 2004] of design science research guidelines below assisting researchers, reviewers, editors, and readers to understand the requirements for effective design science research.

Guideline	Description
Guideline 1: Design as an Artifact	Design science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 3-1: Guideline for Design Science Research [Hevner 2004]

Next, how the project matches the seven guidelines will be presented.

- **Guideline 1 requires producing a viable artifact.**

In the project, the artifact produced is the prototype of city guide covering basic functions such as showing map, locating POIs on map, retrieving information of POIs and so on.

- **Guideline 2 says that objective of design science research is to develop technology-based solutions to important and relevant business problems.**

Service engineering and ubiquitous computing is more and more popular now. The project is given in the context of the project UbiCompForAll. The goal of UbiCompForAll is to provide a solution to enable end users (non-IT professional) compose services according to their personal needs in ubiquitous computing environment. The platform of Android has been chosen to implement mobile services in UbiCompForAll. The project is intended to explore the realization of a city guide on the Android platform. The result of the project can be used as the start point for service composition research, which meets the goal of UbiCompForAll.

- **Guideline 3 requires rigorous and well-executed evaluation methods**

For this project, the evaluation will be performed in four parts. One is platform evaluation that presents how suitable the application is developed on the platform. The second part is general functional evaluation that tests each functional requirement such as users can move around the map and the city guide can retrieve information of the points of interest and so on as said in 4.2 Requirement Specification. Thirdly, we can test if the result of the project can execute scenarios provided in 4.1 Scenarios such as Corporate Group. Lastly, we can evaluate each Non-Functional Requirement such as software extendibility and usability as described in 4.2 Requirement Specification;

- **Guideline 4 requires that design science research can provide clear and verifiable contributions**

There are some existing mobile city guides, as described in 2.1 Existing Mobile City Guides and Similar Solutions. But it is new that the project investigates realization of city guide over Android, which is a new mobile platform with new features such as open source and supporting reuse of components and native access to Google Map infrastructure.

- **Guideline 5 requires rigorous research methods in both construction and evaluation of design artifacts.**

In this project, the methods for construction and evaluation of prototype of city guide are chosen based on the study of various options as exploring existing mobile city guides and analyzing scenarios. Knowledge applied in the project refers to literature

such as introduction of location sensing technologies, wireless internet access and mobile application development technologies.

- **Guideline 6 says that the search for an effective artifact requires using available methods to reach desired ends.**

The result of the project is evaluated in Chapter 6 Evaluation and there are the improvement suggestions given in the section 7.2 Further Work. They serve the iterative process.

- **Guideline 7 says that design science research should be presented not only to technological personnel but also to management audiences.**

Common technical terms will be used (e.g. points of interest is often written with POIs) and keep intended audiences in mind while writing the report.

Chapter 4: Requirement Specification

The part first presents two scenarios based on the scenarios given in [UbiCompForAll 2010]. It helps me understand the functional requirements of the project. Of course, existing mobile city guides described in the *Chapter 2 Background* also help me understand the functional requirement of the project. Finally, I will list the functional requirements in different priorities for implementation and divide them to different groups.

4.1 Scenarios

Scenario is a good approach to generate design idea for new system. It can help people understand better usage of the new system. In the part, I will present two scenarios, which help me produce functional requirements of the project.

4.1.1 Scenario 1: Corporate Group

The Marketing Department of Statoil holds an annual conference in Trondheim. Participants come from different places of the world. The conference finishes on Friday. Some want to spend the weekend visiting the city of Trondheim before they leave. They compose a group, download software of city guide from official website of the local tourist office, and install it in their mobile phones. Then they plan their route for tomorrow visit. They firstly create a group and add members into it. Then they put points of interest that they want to visit into route. And they also create an event reminder for lunch.

The visiting starts in the morning of the weekend. The group members start the city guide on their mobile phones. Then they start “MyTour Plan”, the screen shows the direction from current position to the first points of interest on the map. They go under the help of guide. Moreover, the guide can support audio guide. They find the first point of interest. For knowing more about it, they open the functionality of retrieving information about it. Some of them want to stay more time here but others want to go to the next stop.

So, they are separate. Some arrive at the second stop. They start to visit it and add reviews about it. With the visiting going on, it is time to have lunch. The event reminder alarms, each of them know that it is time to go to the restaurant as planned. The guide shows how to go to the restaurant. But some of them want to delay it because they hope firstly finish the visiting of the current point of interest. So they start functionality of telephone to call other members or send SMS message to others for informing them the delay.

4.1.2 Scenario 2: Family

One family including parent and their two kids plans to travel around the city of Trondheim. They install the application of city guide on their mobile phones. They plan their visit route before starting visit.

In the morning, they start visit, go to sights, and retrieve information about sights. When kids feel hungry and thirsty, parent opens the functionality of “Find Nearest” for finding the restaurant nearby. The guide shows the result and directs them to the restaurant.

Parent’s interest and kids’ interest is often not the same. When the parent are visiting art galleries, kids feel bored and want to be separate with their parent and go to other places they are interested. Parent agrees with their requirement and gives them a mobile phone installing the application of city guide. The parent can locate kids’ location and keep communication with them through the functionality of telephone or SMS message. At the same time, both of them have set an event reminder on their mobile phones for informing them where and when to meet together.

Parent continues visiting and adds reviews for some points of interest. The event reminder alarms for reminding parent and kids to meet together at the proposed location. But now, kids find that they are far away from the location. So they decide to open the functionality of phone, they call Taxi and ask it come to their current position. Finally they ride the Taxi to the location to meet their parent.

4.2 Requirement Specification

After exploring existing mobile city guides and scenarios, I list the functionalities in different priorities. The priorities are given according to the importance of functions for a city guide. For example, the function of showing map is very important for a city guide. The priorities are divided into three different levels: High (H), Medium (M), and Low (L). H is considered the most important for a city guide and implemented firstly, M is less important and implemented after H, L is the least important and considered to implement after the high and medium. Normally, city guide should have basic functions: showing map and showing points of interest (POIs) on map, providing some information about sights or events (e.g. when the museum is open). In addition to them, we can add telephone function, the function of adding review, the function of making tour plan, the function of event reminder and so on. As well, I explore implementation of functionalities as many as possible based on the features of powerful Android platform since investigating realization of a tailorable City Guide application on the Android platform is the main purpose in the project.

4.2.1 Functional Requirements

MAP

ID	Requirement	Priority
1	The city guide can show map of a city.	H
2	The map can be zoomed in and out	H
3	Users can move around the map	H
4	The city guide can show user's current position on the map	H
5	The city guide can show the route between different locations on the map	M
6	The city guide can see other users' positions in the same group on the map	L
7	The map can show points of interest (POIs)	H
8	The map only shows the points of interest users are interested in	M

Table 4-1: Map in Functional Requirements

Information Retrieval

ID	Requirement	Priority
9	The city guide can retrieve information of points of interest	H
10	The city guide can retrieve information of events, which will happen right now.	M
11	The city guide can retrieve information about reviews of points of interest	M
12	Users can select what kind of POIs to show on map (i.e. different classifications. e.g. hotels, museums, restaurants events, and so on).	M

Table 4-2: Information Retrieval in Functional Requirements

Communication

ID	Requirement	Priority
13	The city guide can support communication between users in the same group through telephone or short message.	M

Table 4-3: Communication in Functional Requirements

Review

ID	Requirement	Priority
14	Users can add review about points of interest.	M

Table 4-4: Review in Functional Requirements

Reminder

ID	Requirement	Priority
15	The city guide can support event reminder like lunch time	H

Table 4-5: Reminder in Functional Requirements

Plan tour

ID	Requirement	Priority
16	Users can make a tour plan. (i.e. users can choose any POI and add it into their tour plan.)	M
17	Share tour plan with other users	L

Table 4-6: Plan Tour in Functional Requirements

4.2.2 Non-Functional Requirements

According to the goal of the project, the resulting prototype should be extendible, tailorable, and explore and put existing innovational technologies as many as possible as building blocks in the project. And user interface should be readable, easy to understand, and easy to operate.

ID	Requirement	Priority
18	The resulting prototype should be flexible (extendible and tailorable).	H
19	The user interface should be usable, easy to understand and operate.	H
20	The prototype should combine existing innovational technologies like Google Map as many as possible.	H

Table 4-7: Non-Functional Requirements

Chapter 5: Solution

The chapter presents solution of the project from architecture of solution including general architecture, components diagram, class diagram, and data model, to implementation details of functions.

5.1 General Architecture

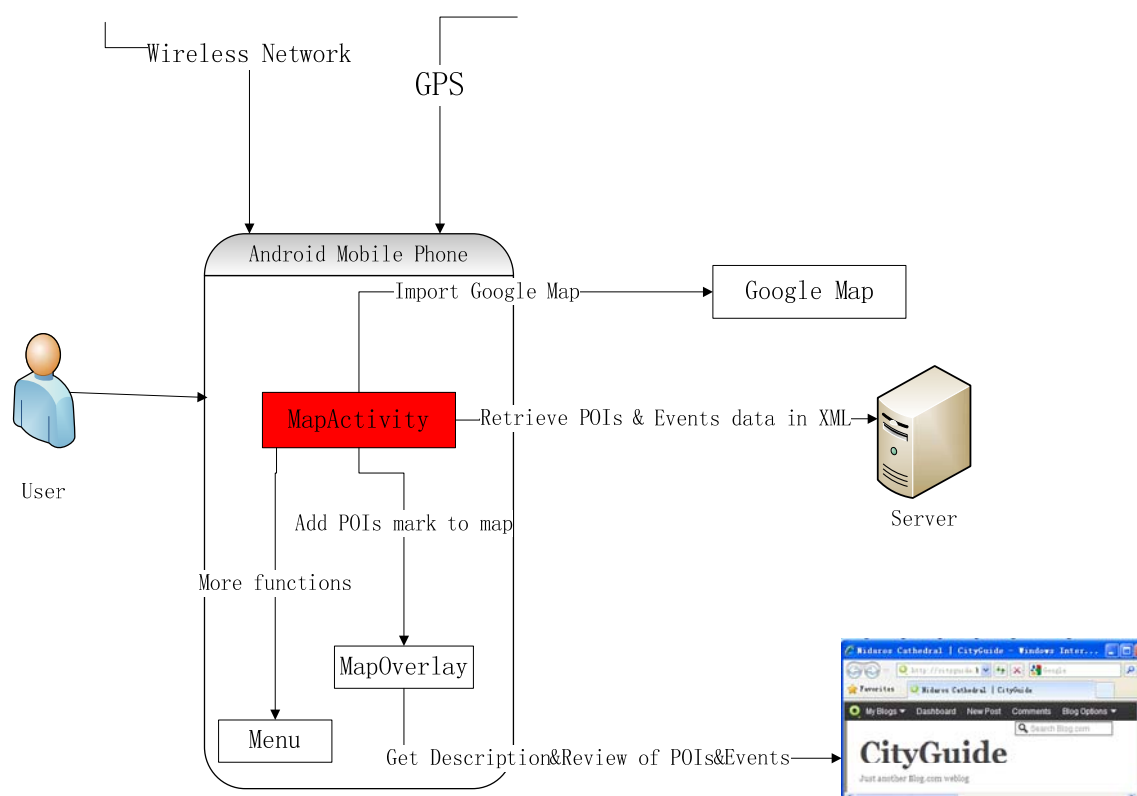


Figure 5-1: General Architecture of Solution

The diagram above presents the general architecture of the prototype. I build the project based on the assumption that users use their Android phones in the environment with wireless network and having the ability of getting GPS data. GPS will be used for automatic localization since android phones are usually equipped with GPS. **MapActivity** in red is the core and the start of application. **MapActivity** imports **Google Map** as the map, and retrieves information of POIs from remote **Server**. **MapActivity** calls **MapOverlay** to add POIs mark to **Google Map**. And **MapActivity** calls **Menu** to present more functions such as reminder, search, and contact and so on.

Google Map is chosen as the map of city guide, I consider it is easy to be implemented on the platform of Android. Both Android and Google Map are released by Google. And we can provide city maps not only for a specific city but also almost all the cities in the world since Google Map show map of the whole world. Therefore, we can provide the service of city guide for many cities only if there is relevant database containing the information of points of interest in cities.

Server is responsible for providing points of interest (POIs) information or event information including location, name, description, category, review, and enabling users add reviews for POIs. The responsible that can feed this information could be tourist office. City guide application visits Server to get this information, which could be provided in the way of XML like below.

```
<sight>
  <name>Nidaros Cathedral</name>
  <link>http://cityguide.blog.com/2010/05/03/nidaros-cathedral/</link>
  <classification>church</classification>
  <latitude>63.4267</latitude>
  <longitude>10.3964</longitude>
</sight>
```

The *name* is the name of POI or event, the *link* is the website on which there is description and review of POIs&Events and users can add reviews. The *classification* is the category of POIs such as museum, hotel and so on. The *latitude* and *longitude* is the location of POI. When the application starts, it retrieves POIs data from XML document as described in Appendix C and stores the data in the local sights database.

Browser showed in the diagram above is triggered from onTap() of **MapOverlay** and is used to open the website, on which there is description and review of POIs and Events, and add review. There is one example of website on <http://cityguide.blog.com/2010/05/03/nidaros-cathedral/> (Email:hanjietongxue@163.com; password: hanjie111).

We also can see screenshots about the example in Figure 6-7: Description of POI about Nidaros Cathedral; Figure 6-8: Review of POI; Figure 6-9: Add Review. The website could be the third party or the one we build on our own in **Server**. It should be better to choose the latter.

5.2 Implementation of Functions

Next, more details of how to implement functions will be presented. Firstly, a component diagram will be presented and it describes all the components used in the solution. Then, I will present implementation of functions of the city guide in the consistent components names as the Component Diagram.

5.2.1 Component Diagram

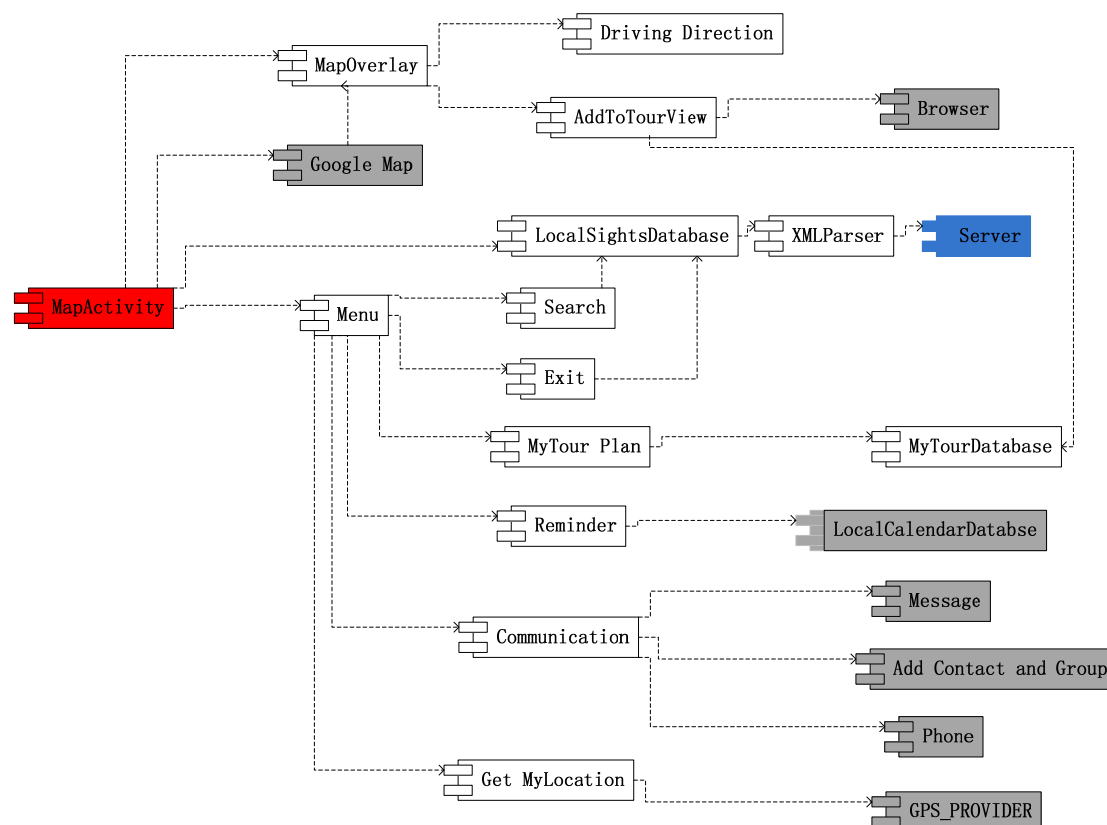


Figure 5-2: Component Diagram

The component diagram above contains all the components in the system. The components in grey already exist. **Server** in blue will not be implemented in the project, but I give the idea of implementation as described in last section and emulate Server for testing the final prototype. The remaining parts will be implemented.

The application starts at the component **MapActivity** in red. **MapActivity** imports **Google Map**, retrieves points of interest (POIs) data from **Server** and store it in **LocalSightsDatabase** that is created when application starts and destroyed when application finishes, calls **MapOverlay** which is overlay object of **Google Map** to draw POIs mark on map or further through **Driving direction** (more details in 5.2.7) is to show route between user's current location and any sight in tour plan on map,

and create **Menu**, which has 6 options: Some of the process described above can be read in 5.2.2 in more details.

1. **Search** is to enable users choose what kind of POIs to show on map such as museum, hotel and so on; it retrieves POIs data from **LocalSightsDatabase**. More details in 5.2.5

2. **Exit** is to exit from the application, at the same time it deletes **LocalSightsDatabase**.

3. **MyTour plan** is to show Tour plan list; it retrieves data from **MyTourDatabase** storing tour plan. We make tour plan starting from **AddToTourView** containing two functions of showing POI description information and review and adding review through **Browser**, and adding the POI into tour plan. POI added will be stored in **MyTourDatabase**. More details in 5.2.6.

4. **Reminder** is to enable users set reminder events, store and retrieve reminder data in **LocalCalendarDatabase** that has been existed and shared by several applications like local calendar application, and Google Calendar. More details in 5.2.4.

5. **Communication** is to enable users add contact, build group and have the function of phone or sending short message. It imports directly the existed application. More details in 5.2.8.

6. **Get MyLocation** is to get user's location data from **GPS_PROVIDER**, which is system service, provided by Android platform and providing the location data of current position. More details in 5.2.3.

The diagram above also show some local database used. Some of them have been existed, and are represented in grey. The rest is under construction.

5.2.2 Map and POIs and Review

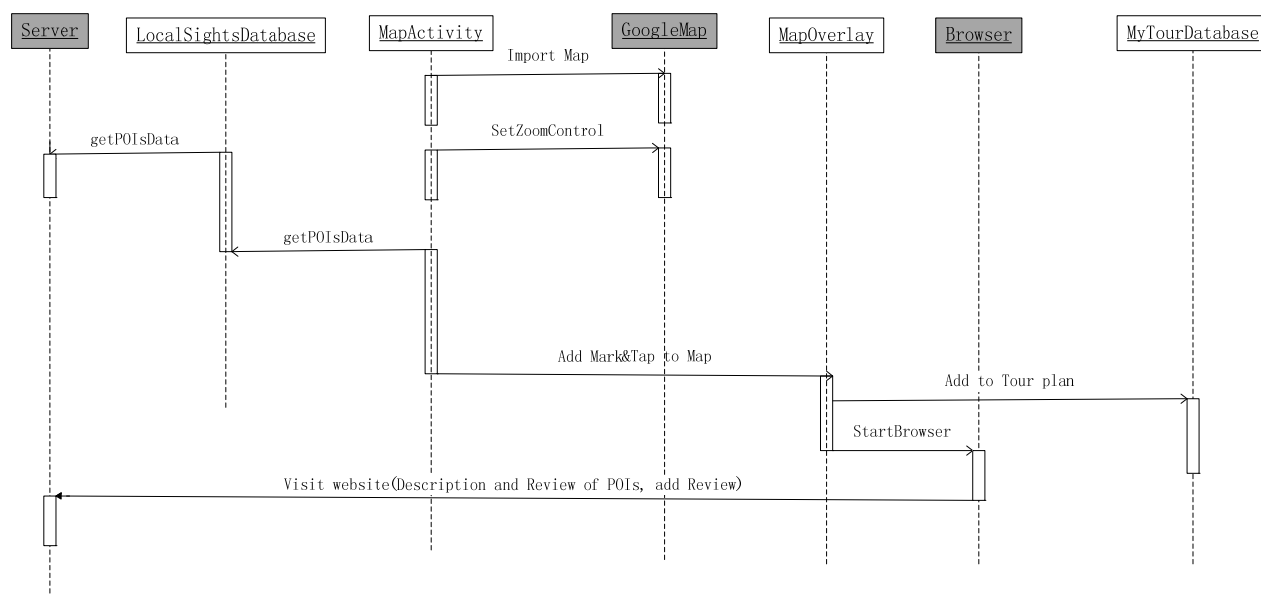


Figure 5-4: Message Sequence Chart of Map and POIs and Review

The Message Sequence chart above shows the process of how to show map, how to add mark to map, how to get POIs data, how to set zoom control, how to show description and review or add review. The grey parts have been existed or will not be implemented in the project.

1. Firstly, **MapActivity** imports **Google Map**.
2. **LocalSightsDatabase** get POIs data from **Server**.
3. Then **MapActivity** gets POIs data from **LocalSightsDatabase** and add POIs mark and tap event command to **MapOverlay** that is the overlay of the map.
4. If users tap the mark (red bubble) of POI, then it asks user to add the POI to tour plan or show description and review about the POI from website, on which users also can add review. There is one example of website on <http://cityguide.blog.com/2010/05/03/nidaros-cathedral/> (Email:hanjietongxue@163.com; password: hanjie111)

5.2.3 Get My Location

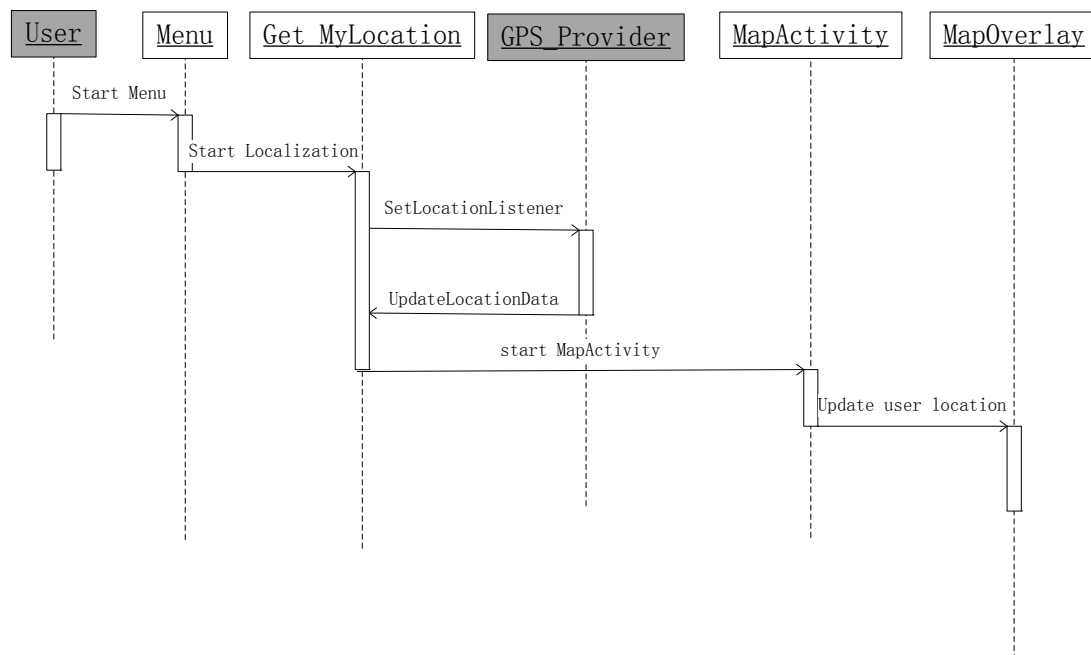


Figure 5-5: Message Sequence Chart of Automatic Localization

The message sequence chart above shows implementation of the functionality of showing user's current position on map. The grey parts have been existed.

1. **Get MyLocation** sets location listener of **GPS_PROVIDER**, which is system service provided by Android platform and providing location data of current position.
2. **GPS_PROVIDER** informs **Get MyLocation** the change of location data in certain frequency, which could be how much time (for example, 1minute) or how much distance (e.g. 10 meters).
3. Then **MapActivity** changes users' current location on map.

5.2.4 Reminder

I create the reminder component on my own, not using the existed local calendar application, because we can not import directly the application into the prototype of city guide. Alternatively, we could choose to use browser to visit Google Calendar to set reminder. I find it is possible through my test. But it leads to one issue that we cannot be sure that each user has Google account. We have to login our Google accounts before setting reminders in Google Calendar. Therefore, I decide to do it on my own. We can use local calendar database for getting and storing reminder event data. The database has been existed and is shared by several applications like local calendar application, Google Calendar.

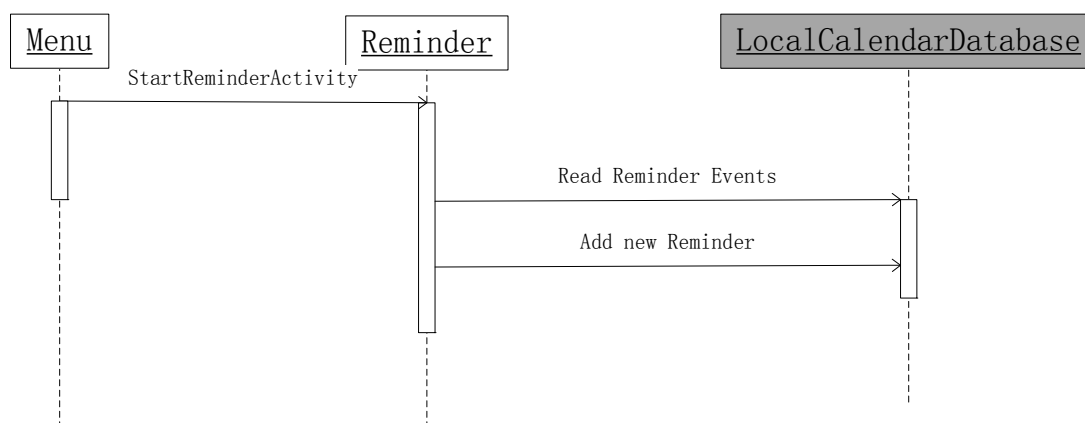


Figure 5-6: Message Sequence Chart of Reminder

The sequence diagram above describes exchange of message to implement the reminder component. The grey part has been existed. From **Menu** to start **Reminder**, **Reminder** gets and stores reminder events data in **local calendar database**, which has been existed and is shared by several applications. The diagram below describes more details about the implementation.

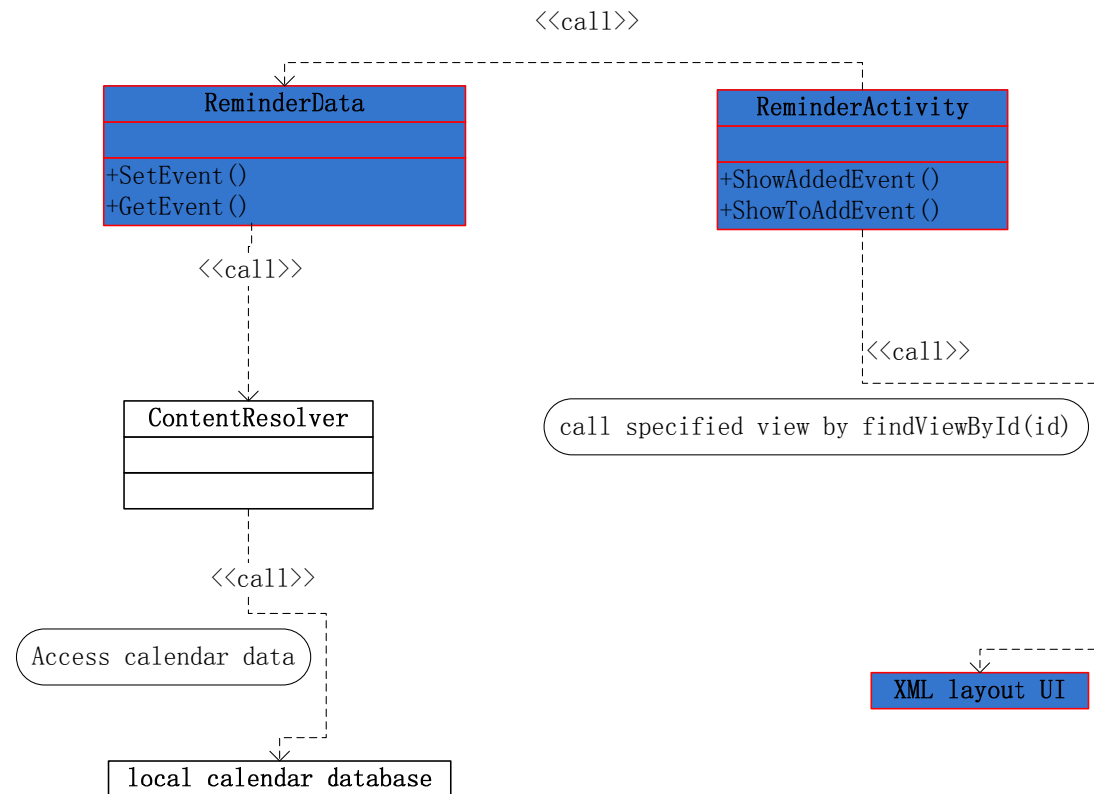


Figure 5-7: Reminder Implementation

ReminderActivity as described in 5.3 Class Diagram is the core part in the reminder solution. It calls **XML layout** for UI showing and calls **ReminderData** for getting data from **local calendar database**. **ContentResolver** is the object which is used for interacting with the content model. It is the API provided by SDK. The parts in blue will be implemented.

5.2.5 Search

Users can choose what kind of POIs to show on map, the categories have museum, church, theatre, hotel, restaurant, old building, event, and other. There is a message sequence chart below showing how to implement the search functionality. From **Menu** to start **Search** component, users choose what kind of POIs, and the **Search** Component gets selected kind of POIs data from **local sight database**, then call **MapActivity** to update map with the selected kind of POIs data.

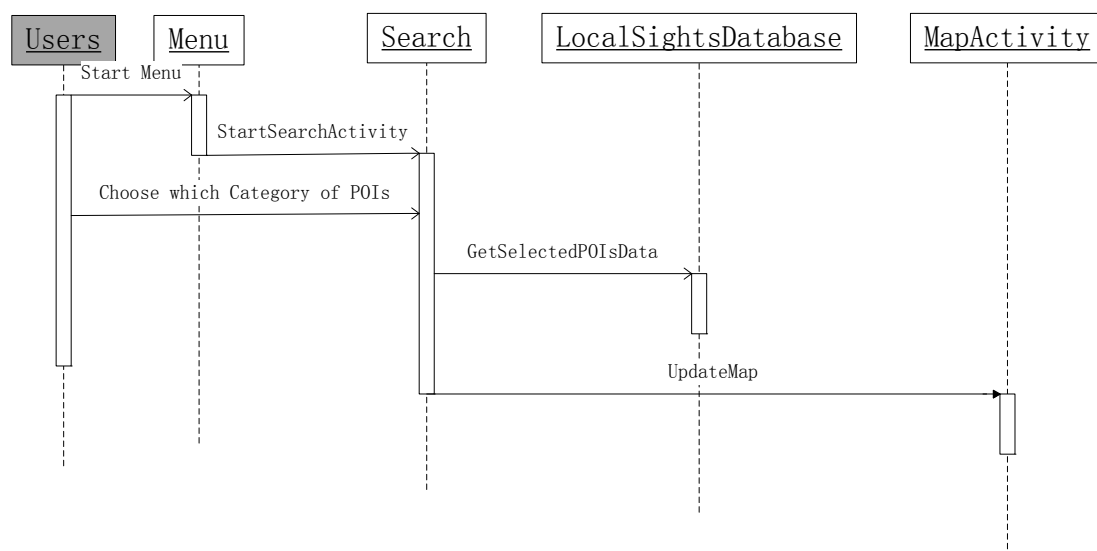


Figure 5-8: Message Sequence Chart of Search

5.2.6 My Tour plan

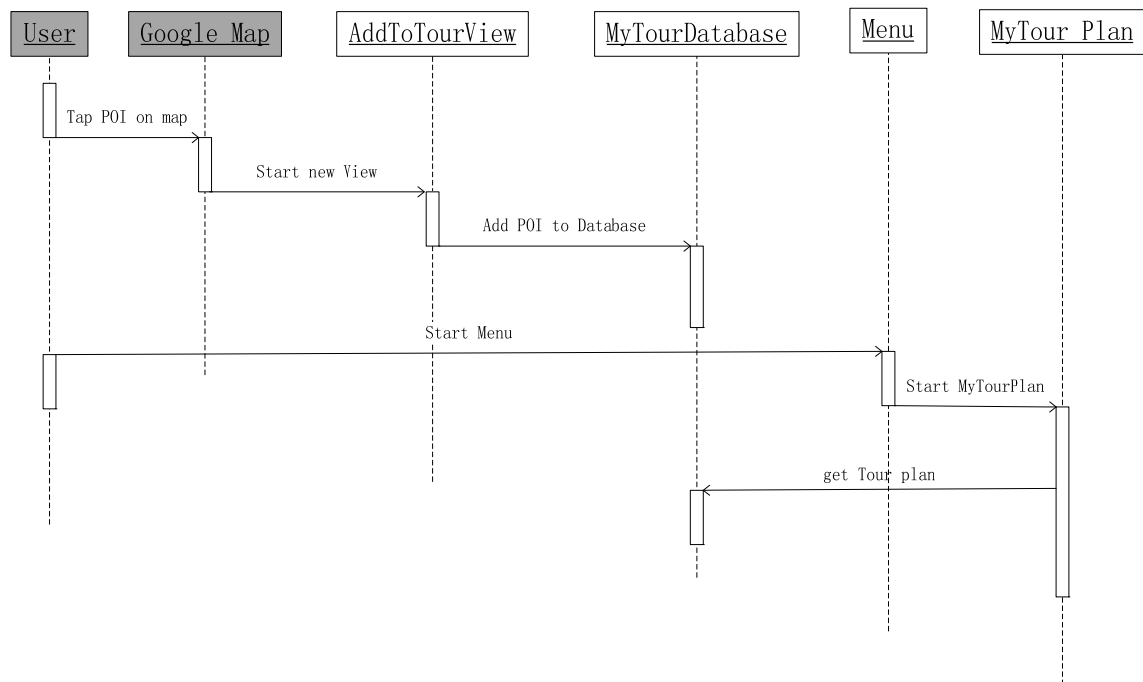


Figure 5-9: Message Sequence Chart of My Tour Plan

The message sequence chart above shows the process of implementing the functionality of making tour plan. The grey parts have been existed. Users tap mark (red bubble) of POI, then interface will be directed to another view **AddToTourView**. Users press button “add to my tour” which means adding selected POI into **My Tour Database**. Then user starts **My Tour Plan** from **Menu**, read the list of sights in the tour plan.

5.2.7 Driving Direction

It meets Functional Requirement 5; the component will show the route from user's current position to any sight in **my tour plan**. The `DrivingDirection` package (`com.google.googlenav.DrivingDirection`) is removed since Android SDK 1.1. There is a different way to finish the function. The message sequence chart below shows how to realize the function. The grey parts have been existed. **User** starts **MyTour Plan** from **Menu** and chooses one sight in tour plan. The selected sight data will be sent to **MapActivity**, which then sends request to **Google Map Sever** with start location data (user's current position) and destination location data (selected POI's position) for getting route data in the way of KML file, then draw it on the map.

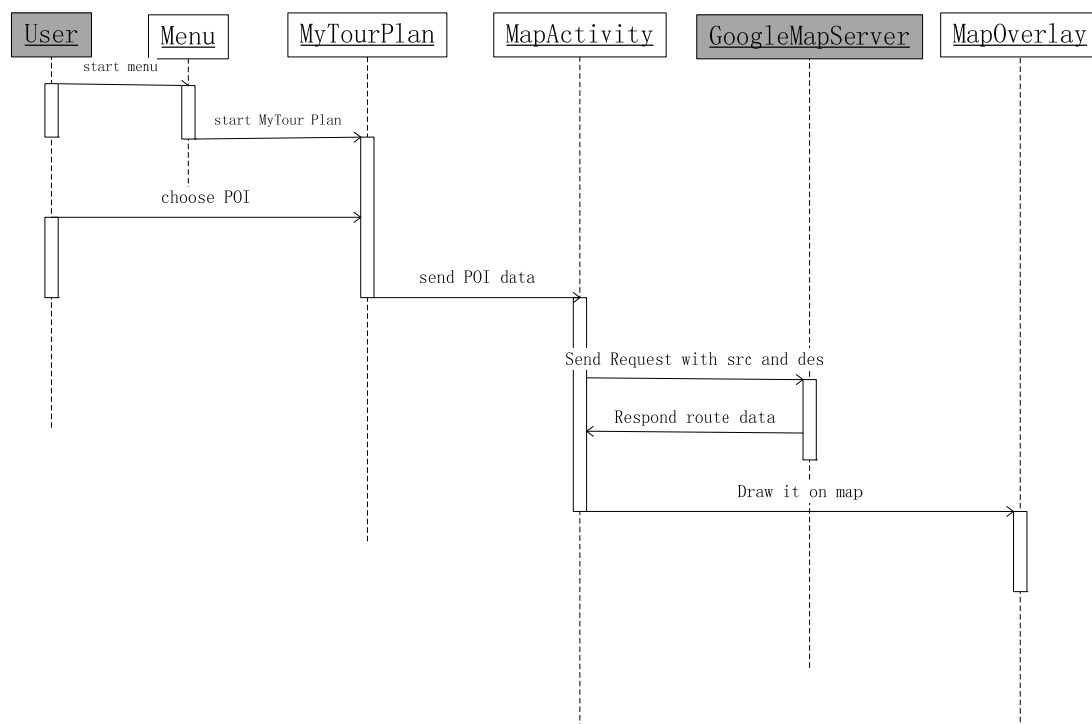


Figure 5-10: Message Sequence Chart of Driving Direction

5.2.8 Communication

In the part, I start Contact application; the code of implementation is showed below:

```
Uri uri=Uri.parse("content://contacts/people");  
Intent intent=new Intent(Intent.ACTION_VIEW,uri);  
startActivity(intent);
```

“content://contacts/people” is uri of the content provider contacts. Contact application will direct user to add new contact, build group, and have a call and send short message.

5.3 Class Diagram

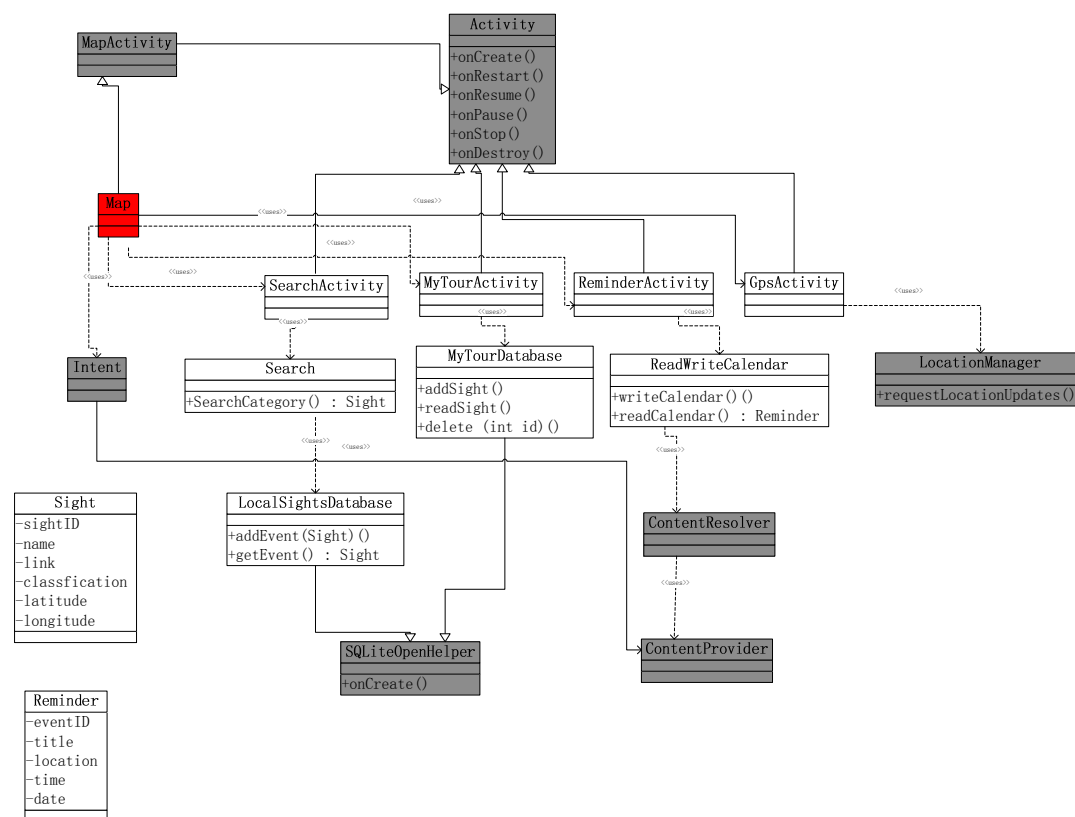


Figure 5-11: Class Diagram of Solution

The figure above presents the class diagram of solution, the grey classes are APIs.

Map in red is the core and the start of the whole application. The most of the source code of **Map** is attached in Appendix E. **Map** extends **MapActivity** from the library `com.google.android.maps`. And **MapActivity** extends **Activity**. **Activity** starts on the method `onCreate()`, and can create a window for you in which you can place your UI with `setContentView(View)`. Then **Map** starts several Activities such as **SearchActivity**: start the function of search, its most source code is attached in Appendix E.

MyTourActivity: start showing my tour plan;

ReminderActivity: start the function of reminder;

GpsActivity: to get user's location, its source code is attached in Appendix E.

Or **Map** can create **Intent** managing interaction between different components such as **Activity** and **ContentProvider** here. For example, we start contact application in the project like below:

```
Uri uri=Uri.parse("content://contacts/people");
Intent intent=new Intent(Intent.ACTION_VIEW,uri);
startActivity(intent);
```

ContentProvider helps retrieve and store data. Here, we use the existing content providers calendar data (`content://calendar/calendars`) and contact data (`content://contacts/people`). However, application does not interact directly with **ContentProvider**, it is through **ContentResolver**. Then **ContentResolver** interacts with **ContentProvider**. We can see the source code of the class `ReadWriteCalendar` in Appendix E.

Android uses SQLite database system. **SQLiteOpenHelper** is a helper class to manage database creation and version management. In the project, we need to build two databases classes extending **SQLiteOpenHelper**. They are **LocalSightDatabase** handling POIs data and **MyTourDatabase** handling data of tour plan. `EventDataSQLHelper.java` is attached in Appendix E and it is for building local sight database. `SQLHandle.java` is also seen in Appendix E and it is for adding sight data into local sight database and get data from local sight database.

The class **Sight** holds sight data and it is implemented in `MapLocation.java` that is seen in Appendix E.

The class **Reminder** holds reminder event data and it is implemented in `ReminderData.java` that can be read in Appendix E.

5.4 Data Model

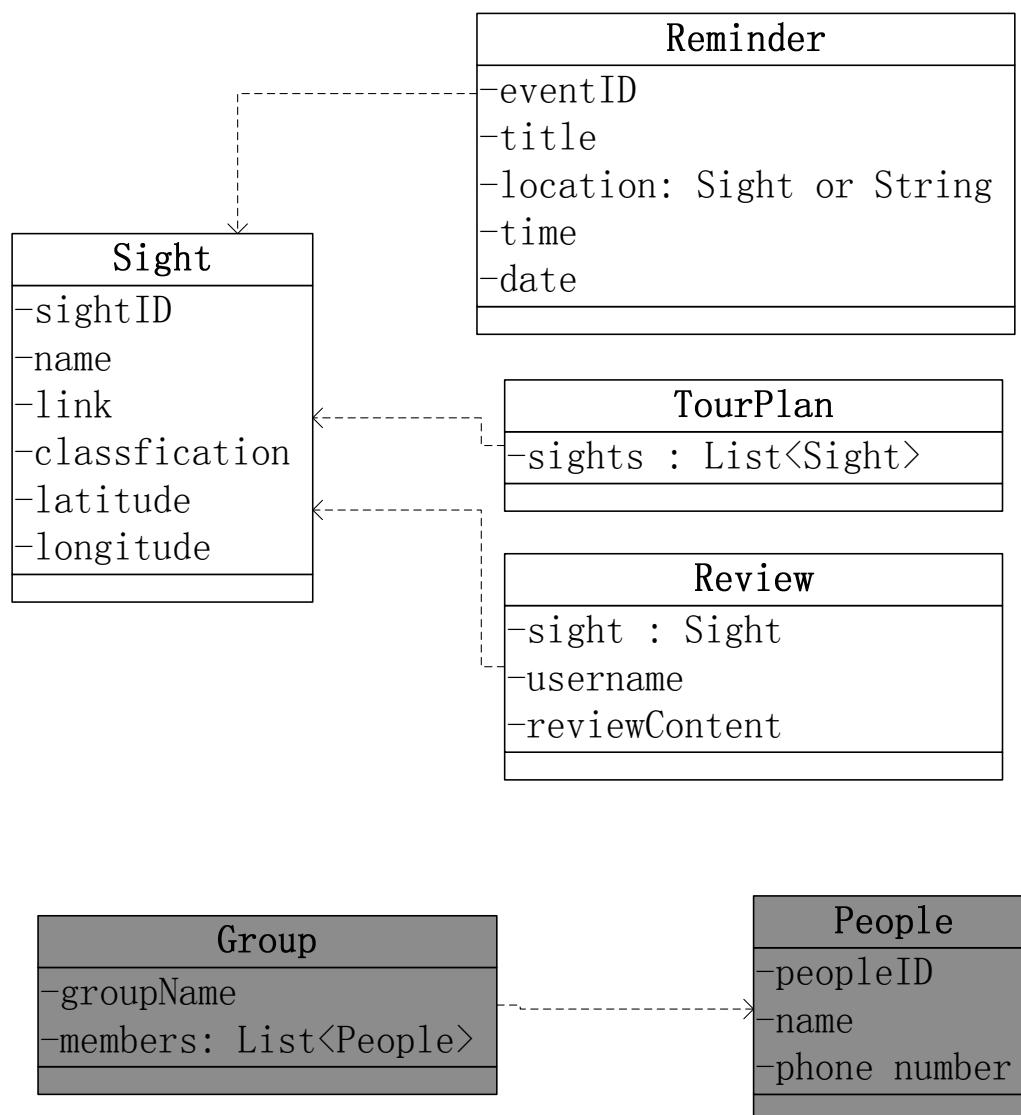


Figure 5-3: Data Model

There is a logical data model represented in Figure 5-3 above and the data model describes the main data used in the project. There are six classes in the model. The grey class means it has been existed.

Sight describes POI information including its **name**, the **link** that is the website address where there is the description of POI, the **classification** that POI belongs to, and **latitude** and **longitude** that stands for location of POI; It will be stored in the database “sights.db”, which is built at the beginning of application and is destroyed

at the finish of application.

Reminder describes reminder event like lunch reminder, which includes event **title**, **location**, **time**, **date**; it stores in existed local calendar database shared by several applications like local Calendar application, outlook express, Google Calendar. The attribute **location** could be one sight. Therefore, its type may be **Sight** or String.

TourPlan shows tour plan with the list of sights; the sights' data type are all **Sight**. The tour plan stores in the database "mytour1.db".

Review describes the review about POIs. It has three attributes. The attribute **sight** represents POI that will be reviewed. The attribute **username** is the name of the person publishing review. The attribute **reviewContent** is the review content.

People present group member's name, phone number. The part directly uses the existed database, so it indicates grey.

Group presents group name and the list of members. Member's data type is **People**.

Chapter 6: Evaluation

The chapter presents evaluation of the project including platform evaluation, functional evaluation and non-functional evaluation, and scenario evaluation.

6.1 Platform Evaluation

The project mainly investigates realization of city guide over Android. I consider that it is important to evaluate what features of Android have been used and how to use them when developing the city guide. We can see that I have used API SQLite to create some databases for storing data like points of interest (POIs) data and tour plan. I applied existing content provider *local calendar database* rather than *local calendar application* into reminder component (It is not possible to import directly *local calendar application* into the prototype of city guide). Retrieving and storing data in ContentProvider is through ContentResolver. As well I applied the existing applications like Browser and Contacts and Phone. Moreover, I used XML layout file to build user interface. And I used API LocationManager to get GPS data from cell phones. Finally, several Activities have been built for presenting different user interfaces that can interact with users. Activity manage lifetime of application.

6.2 General Functional Evaluation

Next, I will give a general functional evaluation. I test each functional requirement. There is the table below showing result of test.

Functional Requirements

MAP

ID	Requirement	Fulfilled?
1	The city guide can show map of a city.	Yes
2	The map can zoom in and out	Yes
3	Users can move around the map	Yes
4	The city guide can show user's current position on the map	Yes
5	The city guide can show the route between different locations on the map	Yes but only show user's current location and sights in tour plan.
6	The city guide can see other users' positions in the same group on the map	No
7	The map can show points of interest	Yes
8	The map only shows points of interest users are interested in	Yes

Table 6-1: Map Evaluation

Information Retrieval

ID	Requirement	Fulfilled?
9	The city guide can retrieve information of points of interest	Yes
10	The city guide can retrieve information of events, which will happen right now.	Yes
11	The city guide can retrieve information about reviews of points of interest	Yes
12	Users can select what kind of POIs to show on map (i.e. different classifications. e.g. hotels, museums, restaurants, events, and so on).	Yes

Table 6-2: Information Retrieval Evaluation

Communication

ID	Requirements	Fulfilled?
13	The city guide can support communication between users in the same group through telephone or short message.	Yes

Table 6-3: Communication Evaluation

Review

ID	Requirements	Fulfilled?
14	Users can add reviews for points of interest.	Yes but we have to provide the website containing the function of adding review.

Table 6-4: Review Evaluation

Reminder

ID	Requirements	Fulfilled?
15	The city guide can support event reminder like lunch time	Yes

Table 6-5: Reminder Evaluation

Plan tour

ID	Requirements	Fulfilled?
16	Users can make a tour plan. (i.e. users can choose any POI and add it into their tour plan.)	Yes
17	Share tour plan with other users	No

Table 6-6: Plan Tour Evaluation

We can see that most of functions are implemented except 6 and 17. Due to the limited time, the two functions have not been done in the project, but here I give my ideas about how to implement them. For 6, it should set listener listening to request of location information from a request side, and then send the location data to the request side. For 17, it can upload tour plan to Server, which enables other users see the tour plan.

6.3 Scenario Evaluation

There is the test performed on the scenario of Corporate Group described in 4.1.1. In the test, we use POIs and Events data in APPENDIX C, which is now uploaded as the file 'test.xml' to <http://folk.ntnu.no/hanjie/>. And POIs and Events information and review are provided in the website <http://cityguide.blog.com>.

Employees from Statoil Company run the application City Guide (current application City Guide is for the city of Trondheim) on HTC Hero with the version 1.5 platform. They login to wireless network. Firstly, they plan their route for tomorrow visit. They start the application City Guide. It shows like Figure 6-1 below: (POIs and Events data are provided as described in Appendix C). We can see showing of Google Map; it is in the Trondheim now. And on the map there are some points of interest such as Kristiansten Fortress, NTNU Gløshaugen and so on, and events like Uka Festival. We also can see zooming bar.

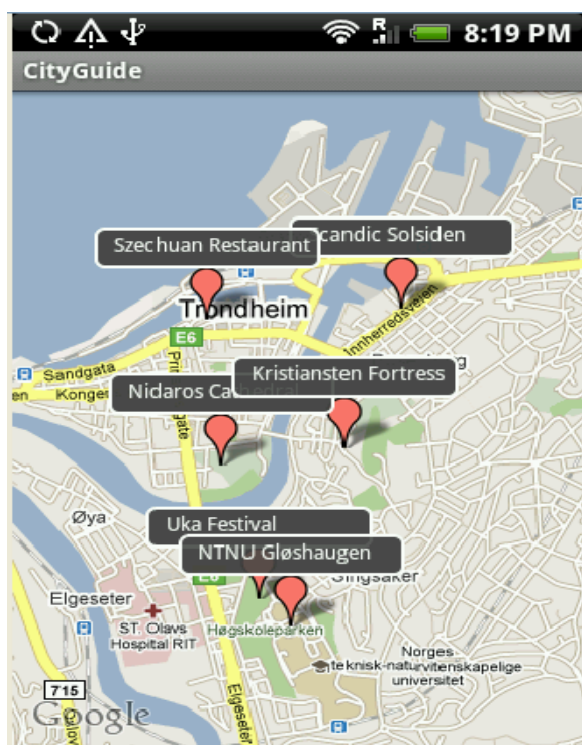


Figure 6-1: Start Application

Then they press the key “menu” on the mobile phone. It shows like Figure 6-3. We can see that there are 6 options in menu. Reminder is for setting reminder events like having lunch, Contact is for adding new contact, building group, calling and sending short message, MyTour plan is for showing tour plan, Search is for choosing what kind of POIs to show on map, Get MyLocation is for showing user’s current position on map, and Exit is for exiting from the application.

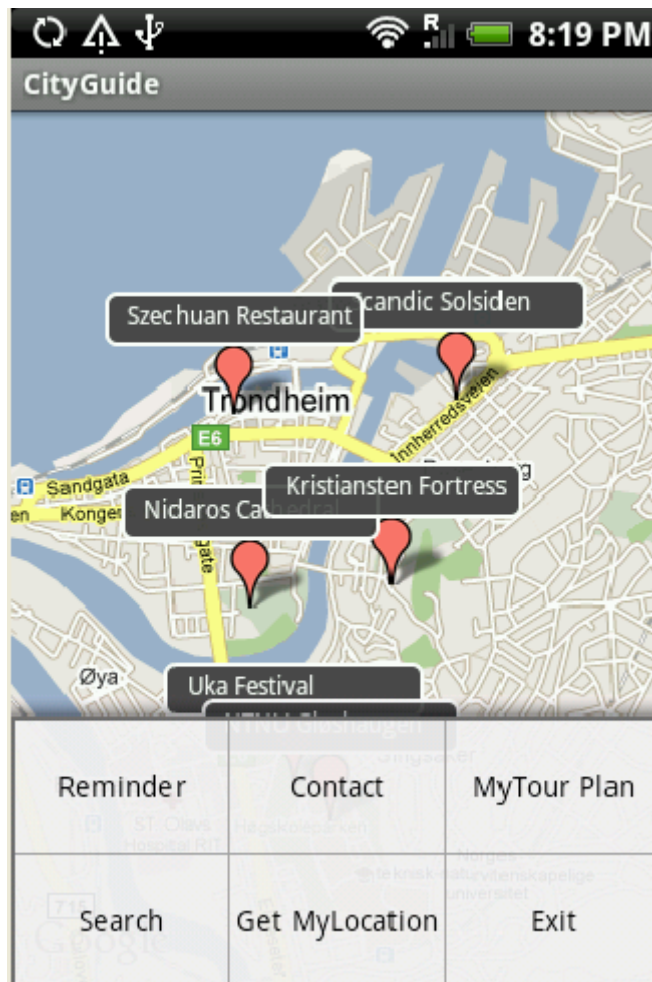


Figure 6-3: Menu

Now they choose Contact in the menu for building group and add each one's contact information. After pressing "Contact", it shows like Figure 6-2. They want to add new contact, so they press "Add contact". It shows like Figure 6-22. We can see the text field of name, and phone number. They move to the icon 'group' at the bottom in Figure 6-2; it shows like Figure 6-23. They press "Add group", it shows like Figure 6-24, we can see the text field of group name. After pressing 'add contact to group', it shows like Figure 6-25. They can choose contact and add it into group.

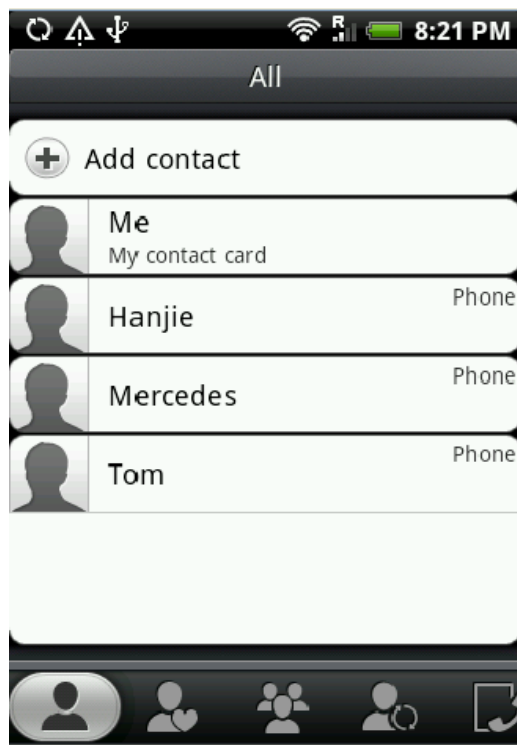


Figure 6-2: Contact

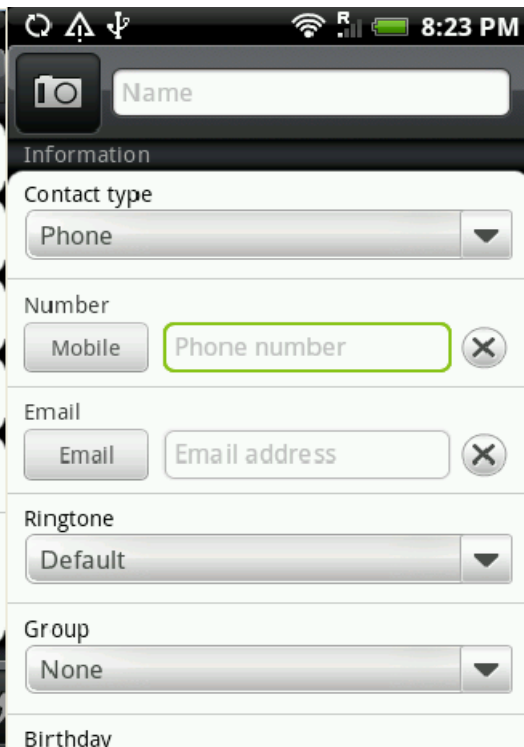


Figure 6-22: New Contact

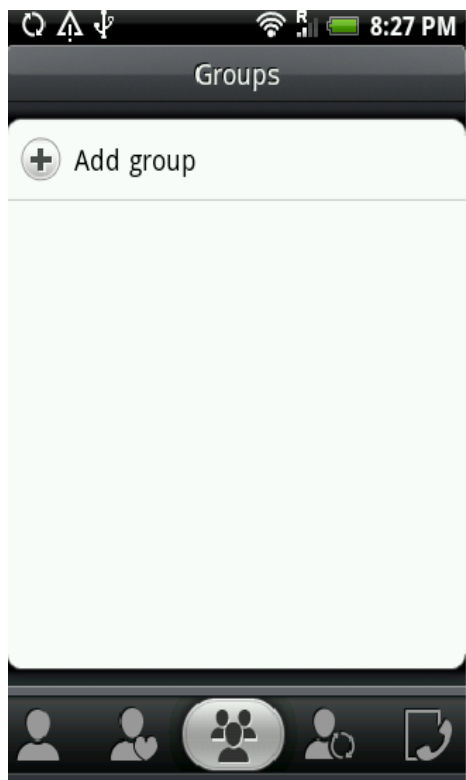


Figure 6-23: Group

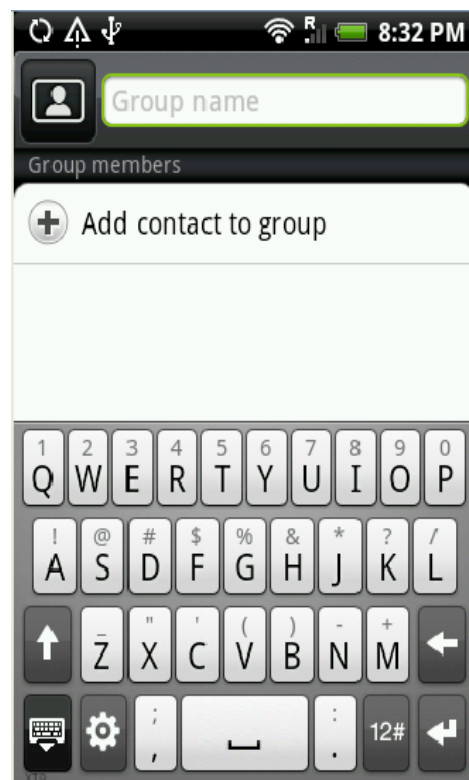


Figure 6-24: new group

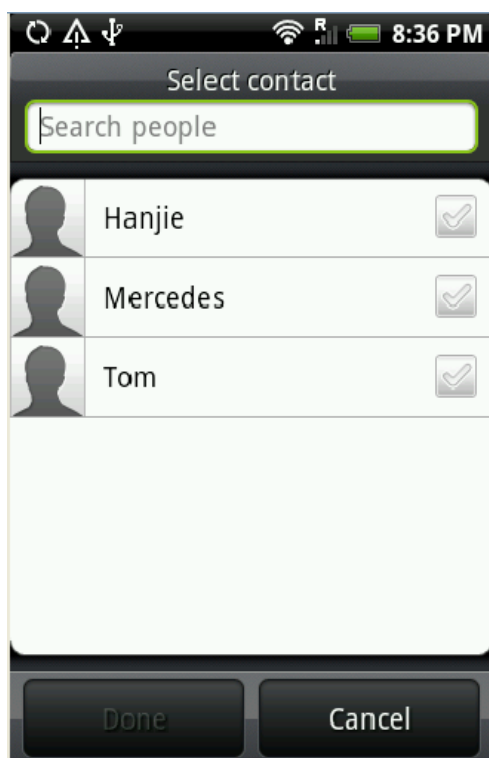


Figure 6-25: Add contact to group.

After add contacts to group, they choose the “Search” in the menu for getting specific sights to show on map. After pressing the option “Search”, it shows like Figure 6-4. We can see that there are several classifications such as church, museum, events and so on. They press the button “church”, it shows like Figure 6-5. We can see ‘Nidaros Cathedral’ showed on map.

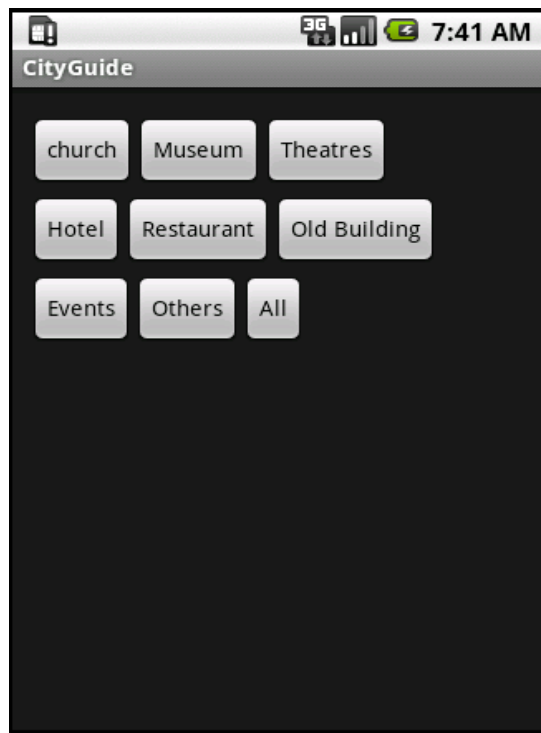


Figure 6-4: Search



Figure 6-5: Result of Search about Church

They want to know more about the POI ‘Nidaros Cathedral’, so they tap red bubble of the POI, then it shows like Figure 6-6. We can see there are two buttons ‘Show Description and Review’ for accessing information about description and review of POIs and ‘Add to My Tour’ for adding the POI to tour plan. They press “Show Description and Review” it shows like Figure 6-7 that is description of POI, Figure 6-8 that is the reviews of POI, and Figure 6-9 for adding review.

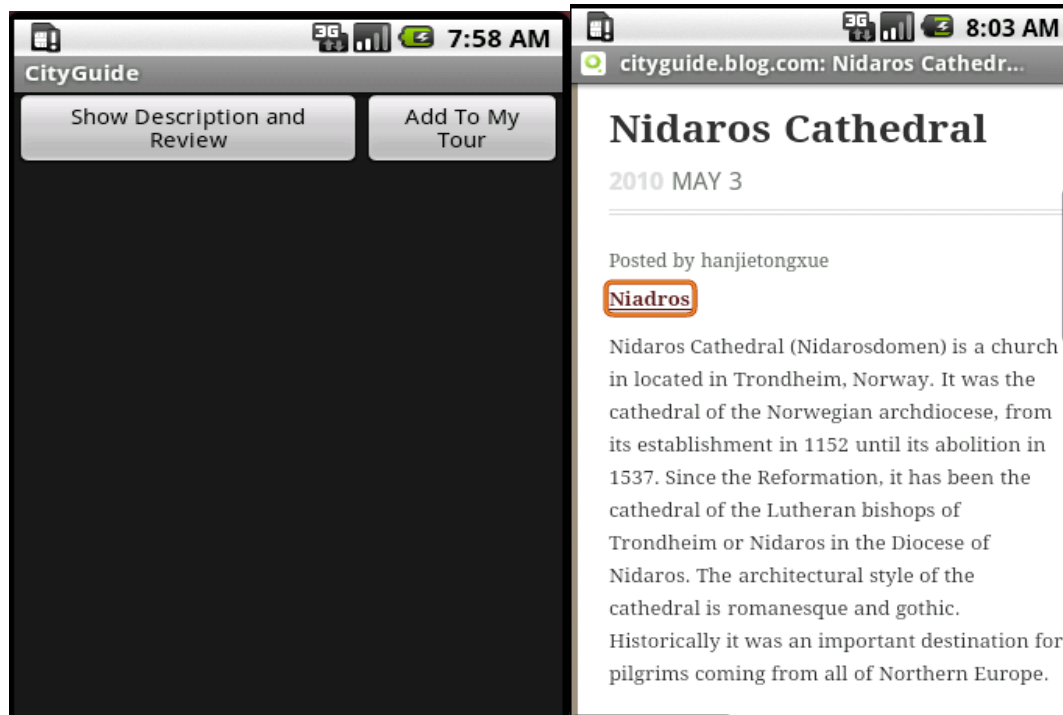


Figure 6-6: Tap on bubble of POI

Figure 6-7: Description of POI about Nidaros Cathedral

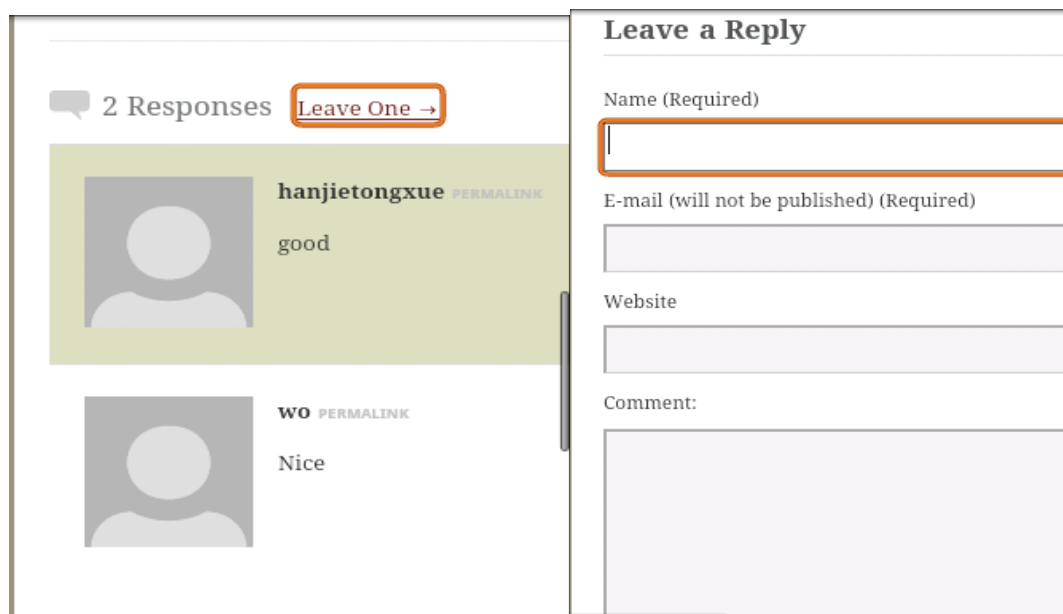


Figure 6-8: Review of POI

Figure 6-9: Add Review

They feel that they should add the POI into Tour plan, so they are back to Figure 6-6 and press "Add to My Tour". They also select other POIs like Kristiansten Fortress, NTNU Gløshaugen and add them into tour plan. Then they start MyTour Plan in the menu as presented in Figure 6-3 and check it. It shows like Figure 6-10 below. We can see that there is a list of sights. They are Nidaros Cathedral, Kristiansten Fortress and NTNU Gløshaugen.

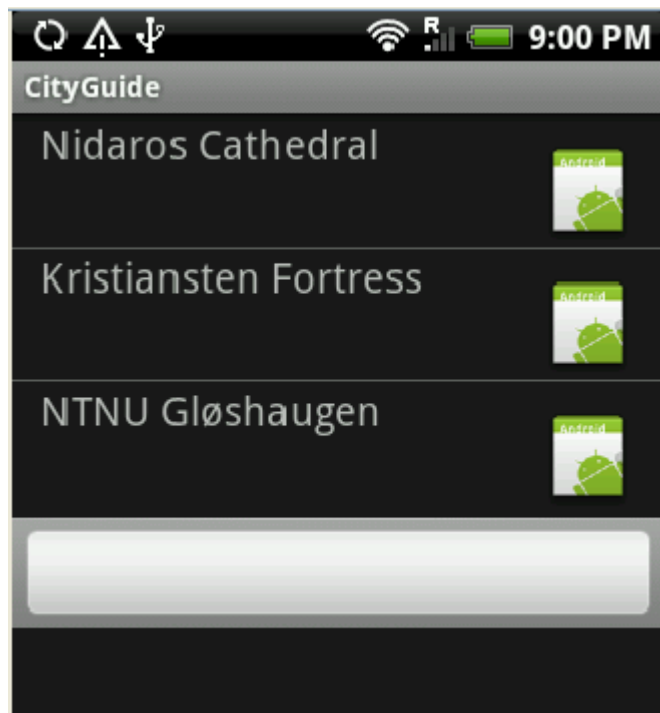


Figure 6-10: My Tour Plan

They also feel that they should add reminder for lunch, so they start reminder in the menu. It shows like Figure 6-11 below. They press the button 'ADD NEW REMINDER'. It shows like Figure 6-12. We can see that there are the text fields of Title and Location, and two buttons for setting time and date. They input title and location, and set time and date as presented in Figure 6-13, 6-14, 6-15, 6-16. Finally, they find that the reminder event they set is added, it is showed in Figure 6-17.

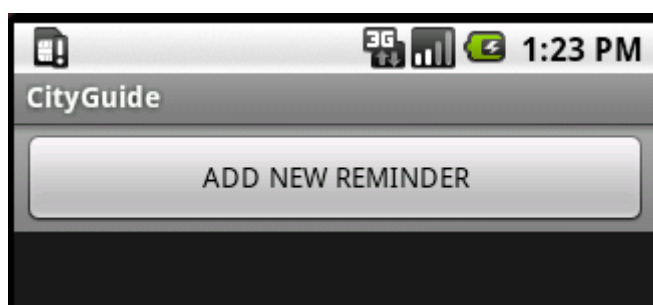


Figure 6-11: Reminder

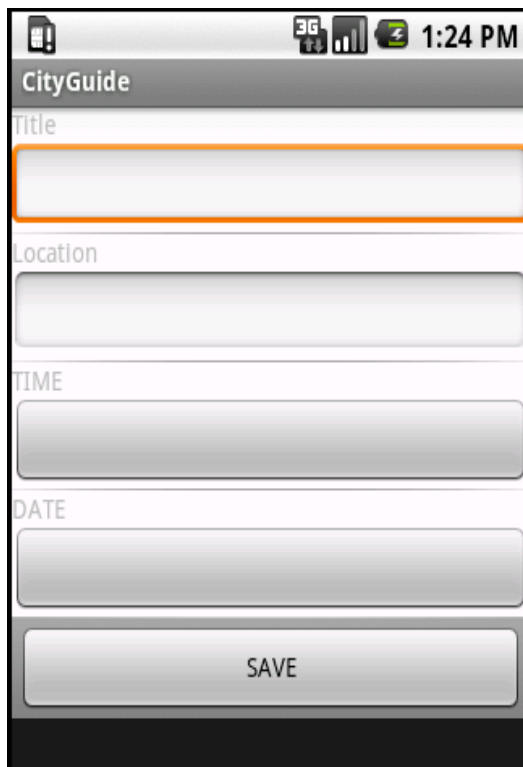


Figure 6-12: Add new Reminder

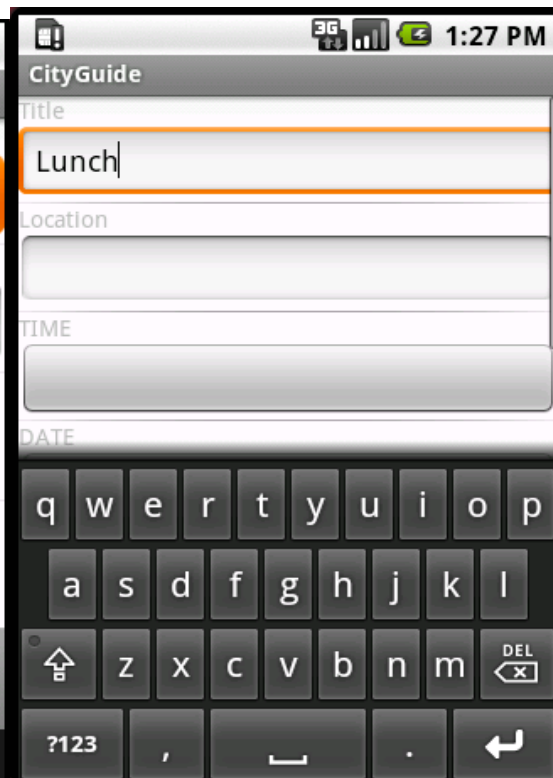


Figure 6-13: Add title of Reminder

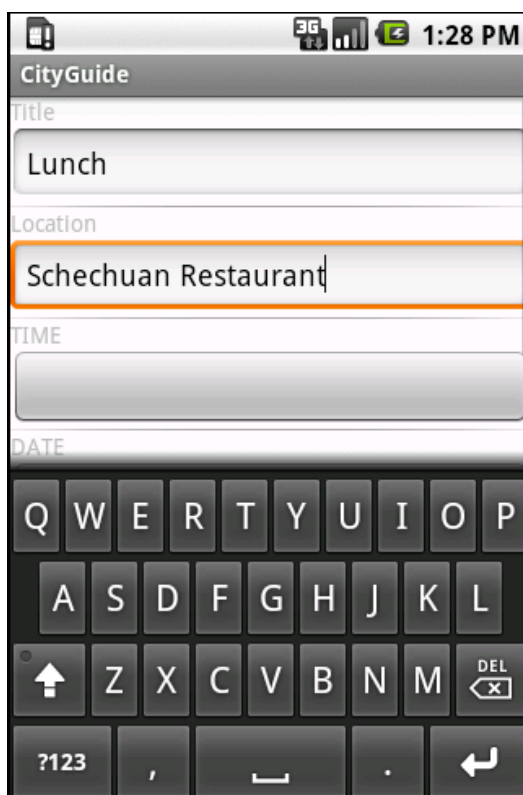


Figure 6-14: Add Location of Reminder

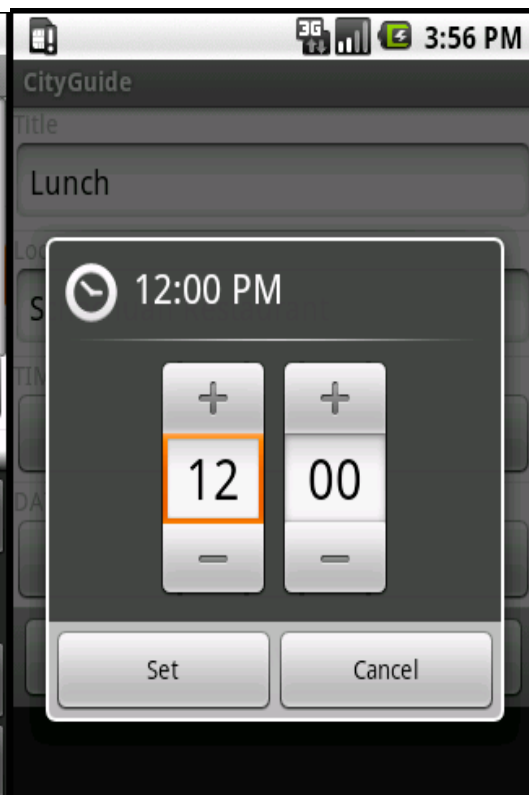


Figure 6-15: Set time of Reminder

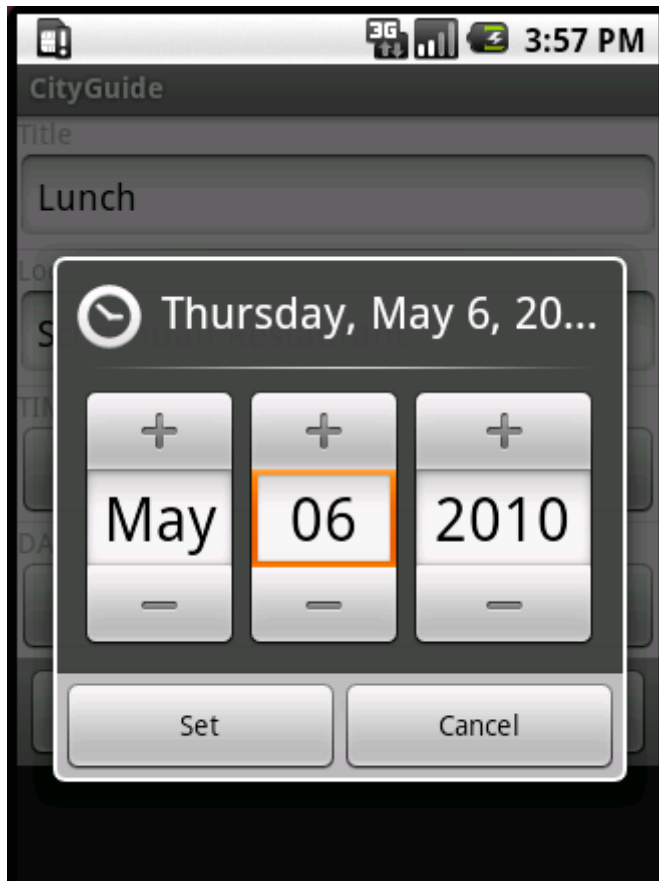


Figure 6-16: Set Date of Reminder

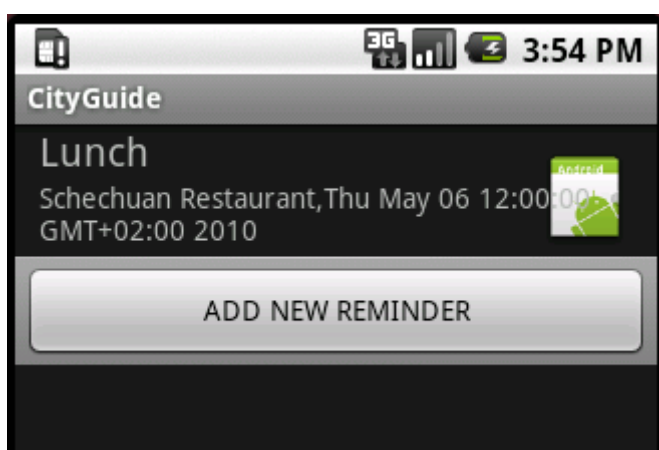


Figure 6-17: Show Reminder Result

In the second day, they start travel the city. They start the application City Guide and they want to get their location, they press “Get MyLocation” in the menu (Figure 6-3). The result shows like Figure 6-26 below for informing fail to get location or Figure 6-18 showing user’s location on map.

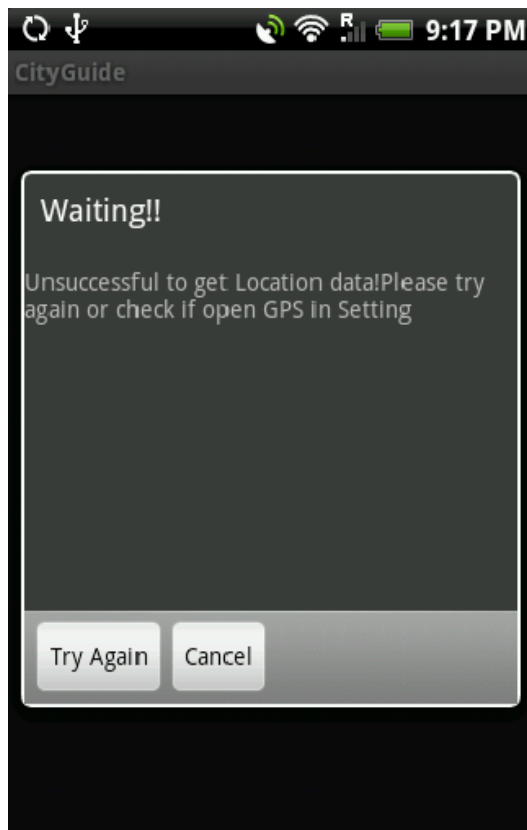


Figure 6-26: Unsuccessful to get Location

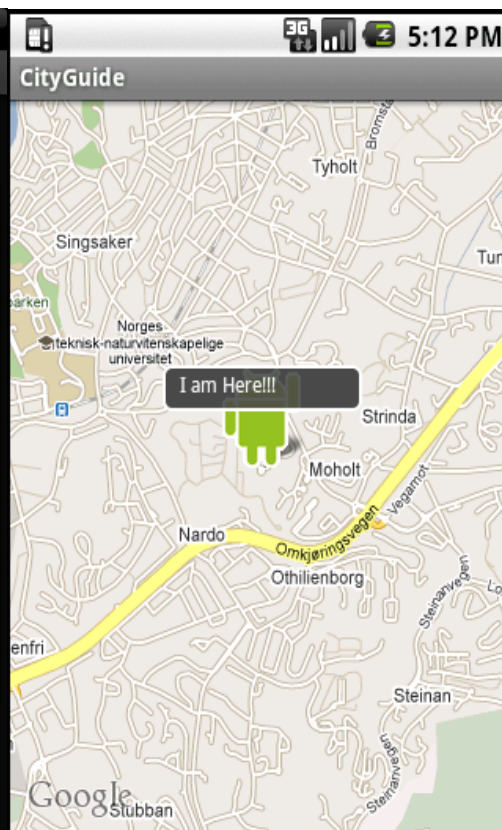


Figure 6-18: Get My Location

They want to go to one sight in the tour plan Figure 6-10. They start MyTour Plan in the menu (Figure 6-3); they choose the sight item Nidaros Cathedral. Then it shows like Figure 6-19. We can see that there are three options “Show Description and Review” for showing the information about the description and review of POI, ‘Show Direction’ for showing route on map between user’s current position and selected sight, and ‘DELETE’ for deleting the sight from tour plan.



Figure 6-19: The result after choosing one Sight in Tour plan

Now they press "Show Direction". The result shows like Figure 6-27 asking users to go to get current location firstly, or like Figure 6-20. We can see that there is the blue line indicating the route from user's current position represented by the blue Android to Nidaros Cathedral.

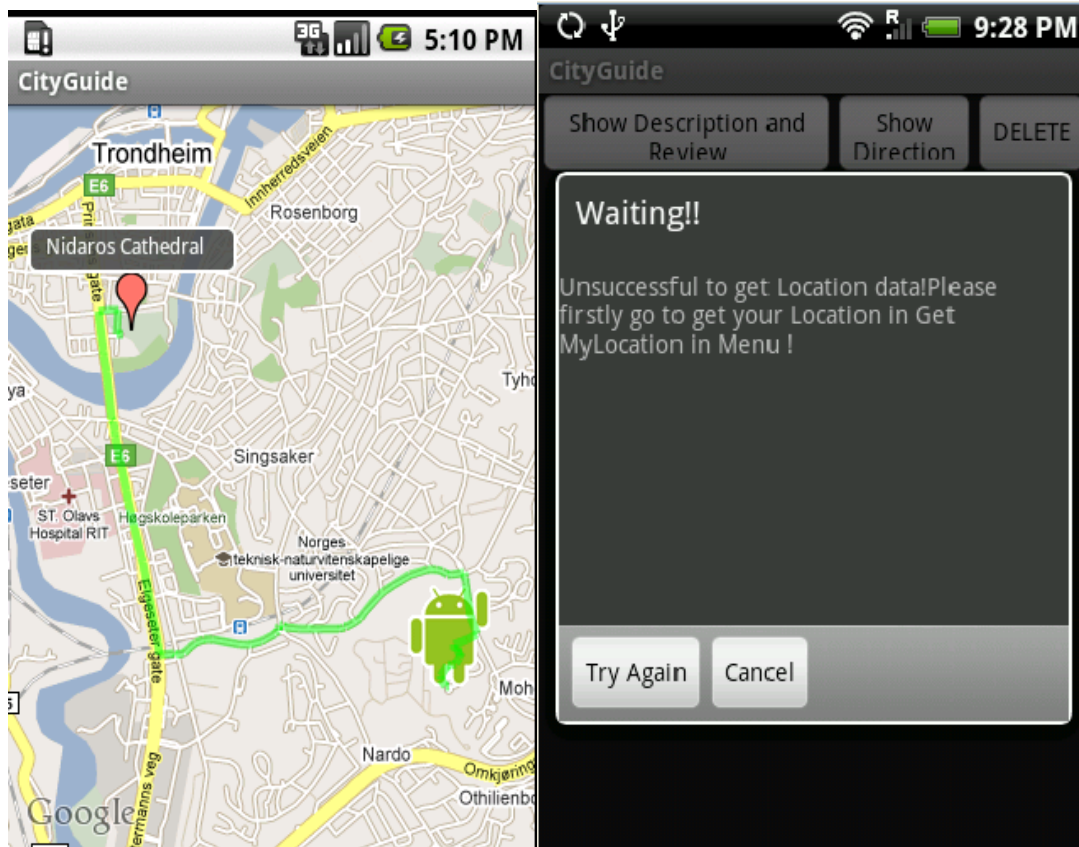


Figure 6-20: Show Direction to one Sight Figure 6-27: Firstly go to get current position

In the noon, the reminder alarms. The alarm voice can be set by the system configuration. Tom is separate from the group after visiting Nidaros Cathedral. The rest in the group want to contact him, they start Contact in the menu and find him; it shows as below:

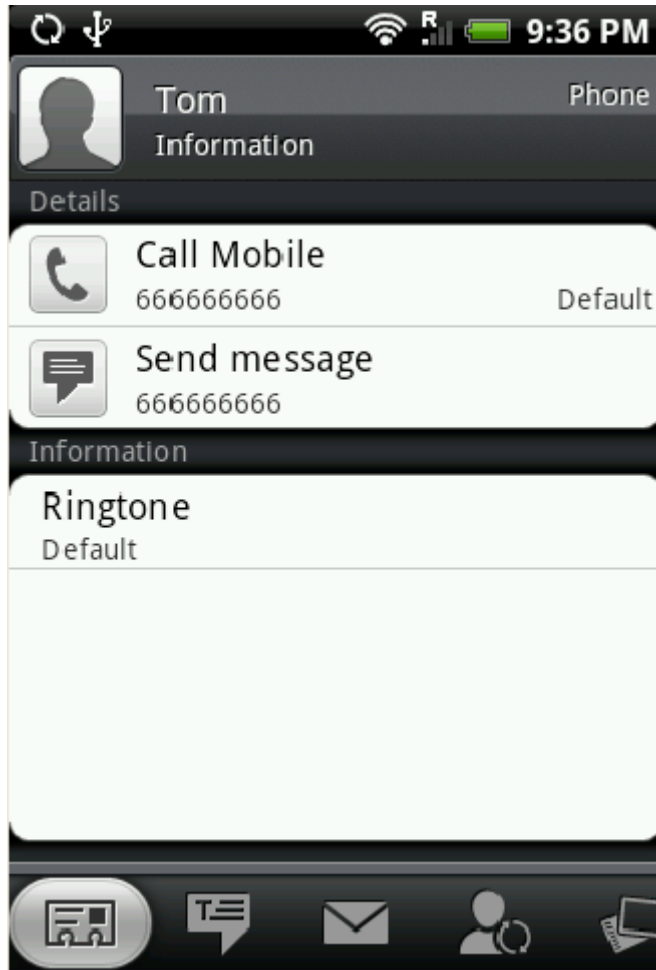


Figure 6-21: Contact with somebody

It looks that the city guide can basically finish all the activities in the scenario description. Therefore, we can say that it is useful.

6.4 Non-Functional Evaluation

6.4.1 Flexibility



The application is easy to extend functions through adding more options in Menu. Each option starts one function. Of course, if you do not need some functions, you can remove them easily from Menu.

The application is easy to change the service from one city to another city since Google Map provides the map of the whole world. We can move the map to next cities by configuring it in the code like below: `mapView.getController().setCenter(new GeoPoint());` GeoPoint represents location.

Of course, we have to enter POIs information of the city into Server side, for example, the tourist office can do it.

The application can get the latest information of POIs since each time it starts it retrieves information from server side. We only need to change data in Server side.

6.4.2 User Interface Friendly

Some icons are used.  is used to mark POIs.  is used to represent user. Menu options are clear to provide more functions to users as described in Figure 6-3. Touch button and text fields for triggering next event are also useful. It should be better if I ask some real users to use the service and get feedback from them for evaluation of the part. Due to limited time, I have not done it this time. It should be done in the future. However, the application adheres to the user interface paradigms used in Android. Therefore I expect that it will be easy for a user that is familiar with Android to use the application.

6.4.3 Combination of Existing Innovational Technologies

In the project, I use the Google Map as the map of city, and integrate Browser to get more information, and build reminder with local calendar database, which can synchronize with Google Calendar when login with the same account like Gmail account. At the same time, I integrate the function of phone and SMS in mobile phones.

Chapter 7: Conclusion and Further work

7.1 Achievements

In the project, I have investigated realization of city guide over Android. The investigation covers basic functions of city guide as described in the section 4.2 Requirement Specification. There are functions such as showing map, showing POIs on map, showing direction, showing user's location on map and so on. Moreover, during the investigation, I try to integrate current innovational technologies as many as possible. Now I put Google Map, Browser, Contacts application, and Phone application into the project. Google Calendar is explored but it does not suit for the project as explained in 5.2.4 Reminder. We are not sure if each user has Google Account. Users have to login to Google Calendar with Google Account if they want to set reminders in Google Calendar. Finally, the prototype is presented. I make a video to show it on

http://www.youtube.com/watch?v=31rB23vjQ6U&feature=PlayList&p=BA6556A0C8EB8EC4&playnext_from=PL&index=1. The video contains 6 files which present how to start application, how to build a group, how to make a tour plan, how to set reminder, how to show direction, and how to contact with others.

And I have used some features of Android when developing the city guide. We can see that I have used API SQLite to create some databases for storing data like points of interest (POIs) data and tour plan. I applied existing content provider like Calendar data into reminder function, and retrieve or store data in ContentProvider is through ContentResolver. As well I applied existing applications like Browser and Contacts and Phone. Moreover, I used XML layout file to build user interface. And I used API LocationManager to get GPS data from cell phones. Finally, several Activities have been built for presenting different user interfaces that can interact with users. Activity manages lifetime of application.

Finally, I have created some XML elements for communicating POIs and Events data such as name, link, classification, latitude and longitude as described in APPENDIX C.

7.2 Further Work

However, there is further work needed to be done.

Firstly, two unimplemented functions (locating others' locations and share tour plan with others) need to be implemented.

Secondly, in the reminder component, it does not distinguish between reminder events related to the application and other reminder events related to other applications sharing the same local calendar database. As a consequence, it also shows reminder events from other applications. The issue should be considered.

Thirdly, we need to consider other location sensing technology like Wi-Fi location since GPS has some shortcomings such as doesn't work indoor, as mentioned in section 2.4 Location Sensing Technologies.

Fourthly, we should consider if the application can contain an off-line way, keep it use even if no Internet access. If so, we have to use offline map and keep LocalSightDatabase mentioned in 5.2.1 in your mobile devices rather than now the database is created when starting the application and destroyed when application is finished. And when Internet can be accessed, the application can update the database.

Fifthly, we should use different marks of POIs & Events to distinguish different classifications rather than using the same mark red bubble in the current solution.

Finally, user evaluation of the city guide prototype should be done in the future.

APPENDIX

A: References

[Android2010]Android;Android—DevGuide,<http://developer.android.com/guide/index.html>, accessed Feb.28th 2010

[AudioTravel,2010]AudioTravel;AudioTravel--MobileTravelGuide,<http://www.audiotravel.com/mobile-travel-guide>, accessed Feb.27th 2010

[AOL,2010]AOL;AOL—CityGuide,<http://mobile.aol.com/aolproducts/cityguide-mobile>, accessed Feb.27th 2010

[Blakstad 2008] Eirik Blakstad; Creating a Mobile City Guide, Project Report, Norwegian University of Science and Technology, 2008

[Cheverst 2000] Cheverst, K., Davies, N., Mitchell, K., Friday, A., Efstratiou, C.; *Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences*. Proceedings of CHI 2000, Netherlands, April 2000, pp 17-24. <http://www.guide.lancs.ac.uk/CHIPaper.pdf>, accessed Feb. 27th 2010.

[GeoPos 2010] GeoPos; *GeoPos NTNU*, <http://dellgeopos.item.ntnu.no/>, accessed May 21st 2010

[Helal 2002] Sumi Helal, Pervasive Java, IEEE Pervasive Computing Magazine, January-March 2002

[Hazas 2004] Mike Hazas, James Scott and John Krumm, *Location-Aware Computing Comes of Age*, IEEE Computer Magazine, February 2004

[Hansen 2007] T. S. Hansen; *A cultural guide for mobile devices*, Project Report, Norwegian University of Science and Technology, 2007

[Hevner 2004] Hevner, March and Jinsoo; *Design Science in Information Systems Research*, MIS Quarterly Vol. 28 2004, pp. 75-105.

[Ibrahim 2008] M. A. Ibrahim; *Utvikling og evaluering av ulike mobile, lokasjonsbaserte tjenester*, Master's thesis, Norwegian University of Science and Technology, 2008

[Kenteris 2007] Kenteris, M., Gavalas, D., Economou, D. (2007); *An innovative*

mobile electronic tourist guide application. Personal and Ubiquitous Computing, Vol. 13, No. 2, February, 2009, p. 103-118. <http://www.springerlink.com/content/887gj235177w7781/>, accessed Feb. 27th, 2010.

[LonelyPlanet,2010]LonelyPlanet;LonelyPlanet--Home,<http://www.lonelyplanet.com/uk>, accessed Feb.27th, 2010

[Nokia6110,2010]Wikipedia;Nokia6110,http://en.wikipedia.org/wiki/Nokia_6110_Navigator, accessed Feb.27th 2010

[Ovimap,2010]Nokia;Ovimap,<http://maps.nokia.com/ovi-services-and-apps/ovi-maps/ovi-maps-main>, accessed Feb. 27th 2010

[Patterson 2003] Cynthia A. Patterson, Richard R. Muntz and Cherri M. Pancake, *Challenges in Location-Aware Computing*, IEEE Pervasive Computing, 03, 2003

[Read 2003] Kris Read and Frank Maurer; *Developing Mobile Wireless Applications*, IEEE Internet Computing, January/February 2003.

[Schmap,2010]Schmap; Schmap—Mobile, <http://www.schmap.com/mobile/>, accessed Feb. 27th 2010

[Sun 2010]Sun; Sun-Java ME, <http://java.sun.com/javame/index.jsp>, accessed Feb.28th 2010

[Traveldodo,2010]Traveldodo;Trvaeldodo--MobileCityGuide,http://www.traveldodo.com/content/free_mobile_city_guide/, accessed Feb.27th 2010

[UbiCompForAll2010]UbiCompForAll;UbiCompForAll-Home, <http://www.ubicompforall.org>, accessed March 17th, 2010.

[UtGuide 2010]UtGuiden; *adressa.no* – *utguiden*, <http://www.adressa.no/kultur/utguiden/>, accessed May 21th, 2010

[Vindigo 2010] Vindigo; *Vindigo: Demo*, <http://www.vindigo.com/demo/demo01.html>, accessed Feb. 26th 2010

[Vibb 2010] Vibb; *Trondheim-guide*, <http://vibb.no/>, accessed May 21th, 2010

[Wayfinder,2010]Wayfinder;Wayfinder–WayfiderNavigatorOnline,<http://wayfinder.com/?id=3991&lang=nl-BE>, accessed Feb. 27th 2010

B: Code Examples

Activity

```
public class Activity extends ApplicationContext {  
  
    protected void onCreate(Bundle savedInstanceState);  
  
        protected void onStart();  
  
        protected void onRestart();  
  
        protected void onResume();  
  
        protected void onPause();  
  
        protected void onStop();  
  
        protected void onDestroy();  
}
```

Intent

Example 1:

```
Intent intent=new Intent(Context context, Activity Class);  
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);  
context.startActivity(intent);
```

Example 2:

```
Intent intent = new Intent();  
intent.setComponent(new ComponentName(package, package.class));  
startActivity(intent);
```

Examp 3(Show web page):

```
Uri uri = Uri.parse("http://google.com");  
Intent it = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(it);
```

Example 4(call phone):

```
Uri uri = Uri.parse("tel:0800000123");  
Intent it = new Intent(Intent.ACTION_DIAL, uri);  
startActivity(it);
```

Example 5(send SMS):

```
Uri uri = Uri.parse("tel:0800000123");  
Intent it = new Intent(Intent.ACTION_VIEW, uri);  
it.putExtra("sms_body", "The SMS text");  
it.setType("vnd.android-dir/mms-sms");  
startActivity(it);
```

C: POIs and Events XML Document Example

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

<!-- generator="FeedCreator 1.7.2" -->

<rss version="2.0">

  <sight>

    <name>Nidaros Cathedral</name>

    <link>http://cityguide.blog.com/2010/05/03/nidaros-cathedral</link>

    <classification>church</classification>

    <latitude>63.4267</latitude>

    <longitude>10.3964</longitude>

  </sight>

  <sight>

    <name>Ringve Museum</name>

    <link>http://cityguide.blog.com</link>

    <classification>museum</classification>

    <latitude>63.2651</latitude>

    <longitude>10.2715</longitude>

  </sight>

  <sight>

    <name>Trondheim Theatre</name>

    <link>http://cityguide.blog.com</link>

    <classification>theatre</classification>

    <latitude>63.425</latitude>
```



```
<longitude>10.35</longitude>

</sight>

<sight>

  <name>Scandic Solsiden</name>

  <link>http://cityguide.blog.com</link>

  <classification>hotel</classification>

  <latitude>63.434512  </latitude>

  <longitude>10.413795</longitude>

</sight>

<sight>

  <name>Szechuan Restaurant</name>

  <link>http://cityguide.blog.com</link>

  <classification>restaurant</classification>

  <latitude>63.434</latitude>

  <longitude>10.395</longitude>

</sight>

<sight>

  <name>Kristiansten Fortress</name>

  <link>http://cityguide.blog.com</link>

  <classification>oldbuilding</classification>

  <latitude>63.4275</latitude>

  <longitude>10.4083</longitude>

</sight>
```

<sight>

<name>Uka Festival</name>

<link><http://cityguide.blog.com></link>

<classification>event</classification>

<latitude>63.42</latitude>

<longitude>10.4</longitude>

</sight>

<sight>

<name>NTNU Gløshaugen</name>

<link><http://cityguide.blog.com></link>

<classification>other</classification>

<latitude>63.418611</latitude>

<longitude>10.403056</longitude>

</sight>

<sight>

<name>Trondheim Airport</name>

<link><http://cityguide.blog.com></link>

<classification>other</classification>

<latitude>63.45000</latitude>

<longitude>10.9333</longitude>

</sight>

</rss>

D: UI Design in XML document

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

E: Relevant Source Code

Map.java

The class Map is the start and the core of the whole application. It imports map, gets POIs data from Server, and create menu.

```
package cityguide.map;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
import cityguide.informationRetrieval.Message;
import cityguide.informationRetrieval.SQLHandle;
import cityguide.informationRetrieval.Search;
import cityguide.informationRetrieval.XmlPullParser;
import cityguide.map.MapLocation;
import cityguide.map.MapLocationOverlay;

public class Map extends MapActivity {

    private static MapLocationOverlay overlay;
```

```
private static MapView mapView;

public static List<MapLocation> mapLocations;

public static List<Message> searches;

private Context context;

public static Map map=new Map();

public SQLHandle SqlHandle=new SQLHandle(this);

public Search searching=new Search(this);


final private int menuReminder = Menu.FIRST;

final private int menuCom = Menu.FIRST+1;

final private int menuMyTour = Menu.FIRST + 2;

final private int search=Menu.FIRST+3;

final private int getMyLocation=Menu.FIRST+4;

final private int exit=Menu.FIRST+5;


public static boolean visit=false;

public static boolean mylocation=false;

private boolean reminder=false;

private boolean contact=false;

private boolean tour=false;

public static boolean add=false;

public static boolean searched=false;


public Map(){
```

```
        map=this;

    }

    // the start of the activity

    public void onCreate(Bundle icle) {

        super.onCreate(icle);

        setContentView(R.layout.main);

        context=this.getApplicationContext();

        mapView = (MapView) findViewById(R.id.mapview1);

        // draw route between user's location and POI.

        if(MyTourActivity.tour){

            double src_lat =GpsActivity.currentLocation.getLatitude();

            double src_long =GpsActivity.currentLocation.getLongitude();

            GeoPoint srcGeoPoint=new GeoPoint((int) (src_lat * 1E6),(int)
                (src_long * 1E6));

            double dest_lat=

            MyTourActivity.tourlist.get(MyTourActivity.arg).getLatitude();

            double dest_long=

            MyTourActivity.tourlist.get(MyTourActivity.arg).getLongitude();

            GeoPoint destGeoPoint =new GeoPoint((int) (dest_lat * 1E6), (int)
            (dest_long * 1E6));

            DrawPath(srcGeoPoint, destGeoPoint, Color.GREEN, mapView);

            mapView.setBuiltInZoomControls(true);
```

```
mapView.getController().animateTo(srcGeoPoint);

mapView.getController().setZoom(15);

}

// update the map

else {

    overlay=new MapLocationOverlay(map);

    updateView(overlay);

}

}

// update map
public static void updateView(MapLocationOverlay overlay){

    mapView.getOverlays().add(overlay);
    mapView.setBuiltInZoomControls(true);
    mapView.getController().setZoom(12);

    // only show user's current position on map
    if(mylocation){

        mapView.getController().setCenter(GpsActivity.currentLocation.getPoint());

    }

    else{
        GeoPoint city=new
        GeoPoint((int)(63.4267*1E6),(int)(10.3964*1E6));
        mapView.getController().setCenter(city);
    }

    mapView.invalidate();

}

// get the list of POIs&Events data including location, category, name,
```

and [website](#) of description and review.

```
public List<MapLocation> getMapLocations() {

    mapLocations = new ArrayList<MapLocation>();

    //only get user's current position
    if(mylocation){

        mapLocations.add(GpsActivity.currentLocation);

    }

    //get the list of all the POIs & Events or part of them after Searching
    else{

        if(!visit){

            XmlPullParser a=new
            XmlPullParser("http://folk.ntnu.no/hanjie/tes
            t.xml");
            List<Message> messages=a.parse();
            for(int i=0;i<messages.size();i++){
                SqlHandle.addEvent(messages.get(i));
            }

            visit=true;

        }

        if(!searched){
            List<Message> sql=SqlHandle.getEvents();
            for(int j=0;j<sql.size();j++)
                mapLocations.add(new MapLocation(sql.get(j)));
        }
        else{
            for(int k=0;k<searches.size();k++)
            {
                mapLocations.add(new
                MapLocation(Search.messages.get(k)));
            }
        }
    }

    return mapLocations;
}
```



```
}

// create menu
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, menuReminder, 0, "Reminder");
    menu.add(0, menuCom, 1, "Contact");
    menu.add(0, menuMyTour, 2, "MyTour Plan");
    menu.add(0, search, 3, "Search");
    menu.add(0, getMyLocation, 4, "Get MyLocation");
    menu.add(0, exit, 5, "Exit");
    return super.onCreateOptionsMenu(menu);
}

// add functions to each menu option
public boolean onOptionsItemSelected(MenuItem item) {

    switch(item.getItemId()) {
        // trigger the function of reminder
        case menuReminder:
            reminder=true;
            Intent intent1=new Intent(context,ReminderList.class);
            intent1.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(intent1);
            break;
        //start the function of phone or message
        case menuCom:
            contact=true;
            Uri uri=Uri.parse("content://contacts/people");
            Intent intent=new Intent(Intent.ACTION_VIEW,uri);
            startActivity(intent);
            break;
        //see the list of my tour plan
        case menuMyTour:
            tour=true;
            Intent intent3=new Intent(context,MyTourActivity.class);
            intent3.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(intent3);
            break;
        // start the function of search in different classifications such
        as museum, church.....
        case search:
            MyTourActivity.tour=false;
```

```
        mylocation=false;
        Intent intent4=new Intent(context,SearchActivity.class);
        intent4.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(intent4);
        break;
        // to get user's current position
    case getMyLocation:
        MyTourActivity.tour=false;
        //visit=true;
        Intent intent5=new Intent(context,GpsActivity.class);
        intent5.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(intent5);
        break;
        // exit the whole application
    case exit:
        context.deleteDatabase("sights.db");
        visit=false;
        searched=false;
        mylocation=false;
        MyTourActivity.tour=false;
        onStop();
        //finish();
        android.os.Process.killProcess(android.os.Process.myPid());
        break;
    }
    return super.onOptionsItemSelected(item);
}

// finish the activity when the activity is not in the foreground
public void onStop() {

    if(reminder|contact|tour|add){
        super.onStop();
    }
    else{
        finish();
        super.onStop();
    }
    tour=false;
    reminder=false;
    contact=false;
    add=false;    }
}
```

MapLocation.java

The class holds location information including name, classification, latitude, longitude and link.

```
package cityguide.map;

import java.net.URL;
import cityguide.informationRetrieval.Message;
import com.google.android.maps.GeoPoint;

public class MapLocation {

    private GeoPoint point;
    private String name;
    private URL link;
    private String link2;
    private String classification;
    double latitude;
    double longitude;

    public MapLocation(double latitude, double longitude){
        point = new GeoPoint((int)(latitude*1e6),(int)(longitude*1e6));
    }

    public MapLocation(Message message){
        this.name = message.getName();
        link=message.getLink();
        point = new
        GeoPoint((int)(message.getLatitude()*1e6),(int)(message.getLon
        gitude()*1e6));
        latitude=message.getLatitude();
        longitude=message.getLongitude();
        classification=message.getClassfication();
    }

    public MapLocation(String name,double latitude1, double longitude1){
        this.name = name;
        latitude=latitude1;
        longitude=longitude1;
        point = new GeoPoint((int)(latitude*1e6),(int)(longitude*1e6));
    }
}
```

```
public MapLocation(String name, double latitude, double longitude,
String link1) {
    this.name = name;
    point = new GeoPoint((int)(latitude*1e6), (int)(longitude*1e6));
    link2=link1;}
public GeoPoint getPoint() {
    return point;
}

public String getName() {
    return name;
}

public String getLink1() {
    return link2;
}

public URL getLink() {
    return link;
}

public double getLongitude() {
    // TODO Auto-generated method stub
    return longitude;
}

public double getLatitude() {
    // TODO Auto-generated method stub
    return latitude;
}

public String getClassfication() {
    // TODO Auto-generated method stub
    return classfication;
}
}
```

Message.java

The class also holds the data of POIs&Events.

```
package cityguide.informationRetrieval;

import java.net.MalformedURLException;
import java.net.URL;

// the class for holding the data of POIs&Events after parsing XML document
public class Message {

    private String name;
    private URL link;
    private String classfication;
    private double latitude;
    private double longitude;
    private int id;

    public void setLink(String link) {
        try {
            this.link = new URL(link);
        } catch (MalformedURLException e) {
            throw new RuntimeException(e);
        }
    }

    public URL getLink(){
        return link;
    }

    public void setName(String nextText) {
        name=nextText;
        // TODO Auto-generated method stub
    }

    public String getName(){
        return name;
    }

    public void setClassfication(String nextText) {
        classfication=nextText;
        // TODO Auto-generated method stub
    }
```

```
    }

    public String getClassfication() {
        return classification;
        // TODO Auto-generated method stub
    }

    public void setLatitude(String nextText) {
        latitude=Double.valueOf(nextText).doubleValue();
        // TODO Auto-generated method stub
    }

    public double getLatitude() {
        return latitude;
        // TODO Auto-generated method stub
    }

    public void setLongitude(String nextText) {
        longitude=Double.valueOf(nextText).doubleValue();
        // TODO Auto-generated method stub
    }

    public double getLongitude() {
        return longitude;
        // TODO Auto-generated method stub
    }

    public void setID(int int1) {
        id=int1;
        // TODO Auto-generated method stub
    }

    public int getID(){
        return id;
    }
}
```

GpsActivity.java

The class is for getting user's current position.

```
package cityguide.map;

import java.util.ArrayList;
import java.util.List;
import com.google.android.maps.MapActivity;
import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class GpsActivity extends MapActivity {

    private LocationManager lm;
    private LocationListener locationListener;
    public static MapLocation currentLocation;
    public static Context context;
    public static GpsActivity map=new GpsActivity();
    public static List<MapLocation> mapLocations;

    public GpsActivity(){
        map=this;
    }

    protected void onDestroy(){
        super.onDestroy();
    }

    protected void onStop(){
        finish();
        super.onStop();
    }

    public void onCreate(Bundle icle) {
```

```
super.onCreate(icle);
lm = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);
locationListener = new MyLocationListener(this);
lm.requestLocationUpdates(
    LocationManager.GPS_PROVIDER,
    0,
    10,
    locationListener);

if (currentLocation!=null){
    Map.mylocation=true;
    Intent intent=new Intent(this,Map.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    startActivity(intent);
}

// if the data of current location is not available.
else{
    //set up dialog
    Dialog dialog = new Dialog(this);
    dialog setContentView(R.layout.gpsdialogue);
    dialog.setTitle("Waiting!!");
    dialog.setCancelable(true);
    //set up text
    TextView text = (TextView) dialog.findViewById(R.id.TextViewgps);
    text.setText("Unsuccessful to get Location data!Please try again
or check if open GPS in Setting");

    //set up button
    Button button = (Button) dialog.findViewById(R.id.Buttongps1);
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            Intent intent4=new Intent(GpsActivity.this,GpsActivity.class);
            intent4.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(intent4);
        }
    });

    Button button2 = (Button) dialog.findViewById(R.id.Buttongps);
    button2.setOnClickListener(new OnClickListener() {
```



```
        public void onClick(View v) {
            finish();
        }});
    //now that the dialog is set up, it's time to show it
    dialog.show();
}

}

public List<MapLocation> getMapLocations() {

    mapLocations = new ArrayList<MapLocation>();
    mapLocations.add(currentLocation);
    return mapLocations;
}

// set the listener for change of location
public static class MyLocationListener implements LocationListener
{

    public MyLocationListener(Context context1) {
        context=context1;
    }

    @Override
    public void onLocationChanged(Location loc) {

        if (loc != null){
            currentLocation=new MapLocation("I am
            here!!!!",loc.getLatitude(), loc.getLongitude());
        }
    }
}}
```

SearchActivity.java

The class is for implementing the function of search.

```
package cityguide.map;

import cityguide.informationRetrieval.Search;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class SearchActivity extends Activity {

    public void onDestroy(){
        super.onDestroy();
    }
    public void onStop()
    {
        super.onStop();
        finish();
    }
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icle) {

        super.onCreate(icle);
        setContentView(R.layout.search);
        // read the class Search below
        final Search search=new Search(this);

        Button tutorial1Button =
            (Button)findViewById(R.id.button_view_tutorial_1);
        tutorial1Button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                Map.searched=true;
                Map.searches=search.SearchCategory("church");
                Intent intent4=new Intent(SearchActivity.this,Map.class);
                intent4.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent4);
            }
        });

        Button tutorial2Button =
```

```
(Button)findViewById(R.id.button_view_tutorial_2);
tutorial2Button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Map.searches=search.SearchCategory("museum");
        Map.searched=true;
        Intent intent4=new Intent(SearchActivity.this,Map.class);
            intent4.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent4);
        });
    //other categories are omitted here...
}
```

Search.java

The class is for getting the list of POIs as the searched result.

```
package cityguide.informationRetrieval;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class Search extends Activity{

    public static ArrayList<Message> messages;
    private SQLiteOpenHelper eventsData;
    public Context context;

    public Search(Context context1){
        context=context1;
        eventsData = new EventDataSQLHelper(context);
    }
    public List<Message> SearchCategory(String category){

        messages=new ArrayList<Message>();
        SQLiteDatabase db = eventsData.getReadableDatabase();
        Cursor cursor = db.query(EventDataSQLHelper.TABLE, null,
null, null, null,null, null);
```

```
startManagingCursor(cursor);

while (cursor.moveToNext()) {

    if(category.equals("all")){

        Message message=new Message();
        message.setName(cursor.getString(1));
        message.setLink(cursor.getString(2));
        message.setClassification(cursor.getString(3));
        message.setLatitude(cursor.getString(4));
        message.setLongitude(cursor.getString(5));
        messages.add(message);

    }

    else{

        if(cursor.getString(3).equals(category)){

            Message message=new Message();
            message.setName(cursor.getString(1));
            message.setLink(cursor.getString(2));
            message.setClassification(cursor.getString(3));
            message.setLatitude(cursor.getString(4));
            message.setLongitude(cursor.getString(5));
            messages.add(message);

        }

    }

}

return messages;

}
```

ReadWriteCalendar.java

The class is for getting reminder events or writing reminder events from or to local calendar database

```
package cityguide.reminder;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import cityguide.map.ReminderActivity;
import android.app.Activity;
import android.content.ContentResolver;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.text.format.DateUtils;

public class ReadWriteCalendar extends Activity{

    private static int mHour;
    private static int mMinute;
    private static int mDay;
    private static int mMonth;
    private static int mYear;

    public static void writeCalendar(Context context){

        ContentResolver contentResolver=context.getContentResolver();

        contentResolver.query(Uri.parse("content://calendar/calendars"),
        (new String[] { "_id", "displayName", "selected" } ), null, null, null);

        ContentValues event = new ContentValues();
        event.put("calendar_id", "1");
        event.put("title",ReminderActivity.title.getText().toString());
        event.put("description", "");
        event.put("eventLocation",
        ReminderActivity.location.getText().toString());
        event.put("visibility", 3);
        event.put("eventStatus", 1);
        event.put("allDay", 0);
```

```
Date startTime =new Date(mYear-1900,mMonth,mDay,mHour,mMinute);
event.put("dtstart",startTime.getTime());
event.put("hasAlarm", 1);

final ContentResolver cr = context.getContentResolver();
Uri newEvent = cr.insert( Uri.parse( "content://calendar/events" ),
event );
long id = Long.parseLong( newEvent.getLastPathSegment() );

if( newEvent != null ) {
    ContentValues values = new ContentValues();
    values.put( "event_id", id );
    values.put( "method", 1 );
    values.put( "minutes",0 );
    cr.insert( Uri.parse( "content://calendar/reminders" ), values );

    values = new ContentValues();
    values.put( "event_id", id );
    values.put( "begin", startTime.getTime());
    //values.put( "end", item.getEndTime() );
    values.put( "alarmTime", startTime.getTime() );
    values.put( "state", 0 );
    values.put( "minutes", 0 );
    cr.insert( Uri.parse( "content://calendar/calendar_alerts" ),
    values );
}

}

public static List<ReminderData> readCalendar(Context context) {

    ContentResolver contentResolver=context.getContentResolver();
    contentResolver= context.getContentResolver();

    // display all the events from the previous week to the end of next week
    Uri.Builder builder =
    Uri.parse("content://calendar/instances/when").buildUpon();

    long now = new Date().getTime();
    ContentUris.appendId(builder, now - DateUtils.WEEK_IN_MILLIS);
    ContentUris.appendId(builder, now + DateUtils.WEEK_IN_MILLIS);
```

```
Cursor eventCursor = contentResolver.query(builder.build(),
new String[] { "title", "begin", "end", "allDay", "eventLocation"},
"Calendar_ID="+ "1", null, "startDay ASC, startMinute ASC");
    List<ReminderData> reminderlist=new ArrayList<ReminderData>();

    while (eventCursor.moveToNext()) {

        final String title = eventCursor.getString(0);
        final Date begin = new Date(eventCursor.getLong(1));
        final Date end = new Date(eventCursor.getLong(2));
        final Boolean allDay =
eventCursor.getString(3).equals("0");
        final String location=eventCursor.getString(4);
        ReminderData reminderItem=new
ReminderData(title,begin,end,location);
        reminderlist.add(reminderItem);

    }
    return reminderlist;
}

void delete(int arg, Context context){
    ContentResolver contentResolver=context.getContentResolver();
    contentResolver= context.getContentResolver();
    Uri.Builder builder =
Uri.parse("content://calendar/instances/when").buildUpon();
    long now = new Date().getTime();
    ContentUris.appendId(builder, now - DateUtils.WEEK_IN_MILLIS);
    ContentUris.appendId(builder, now + DateUtils.WEEK_IN_MILLIS);

    Cursor eventCursor = contentResolver.query(builder.build(),
        new String[] { "title", "begin", "end",
"allDay", "eventLocation"}, "Calendar_ID="+ "1",
        null,
        "startDay ASC, startMinute ASC");
}

public static void setHour(int hourOfDay) {
    mHour=hourOfDay;
    // TODO Auto-generated method stub
}

public static void setMinute(int Minute) {
    mMinute=Minute;
    // TODO Auto-generated method stub
}
```

```
    }  
    public static void setDay(int day) {  
        // TODO Auto-generated method stub  
        mDay=day;  
    }  
    public static void setMonth(int month) {  
        // TODO Auto-generated method stub  
        mMonth=month;  
    }  
    public static void setYear(int year) {  
        // TODO Auto-generated method stub  
        mYear=year;  
    }  
}}
```

ReminderData.java

The class holds reminder event data.

```
package cityguide.reminder;  
  
import java.util.Date;  
  
public class ReminderData {  
  
    String title;  
    String description;  
    String location;  
    Date startTime;  
    long endTime;  
    boolean hasAlarm;  
    boolean allDay;  
  
    public ReminderData(String title2, Date begin, Date end,String  
location1) {  
        title=title2;  
        startTime=begin;  
        endTime=end.getTime();  
        location=location1;  
    }  
    public void setTitle(String title1){
```



```
        title=title1;
    }
    public String getTitle(){
        return title;
    }
    public String getLoation(){
        return location;
    }
    public Date getDate(){
        return new Date();
    }

    public void setDescription(String description1){
        description=description1;
    }
    public String getDescription(){
        return description;
    }

    public void setStartTime(Date startTime1){
        startTime=startTime1;
    }
    public Date getStartTime(){
        return startTime;
    }

    public void setEndTime(long endTime1){
        endTime=endTime1;
    }
    public long getEndTime(){
        return endTime;
    }

    public void setAlarm(boolean hasAlarm1){
        hasAlarm=hasAlarm1;
    }
    public boolean getAlarm(){
        return hasAlarm;
    }

    public void setAllday(boolean allday){
        allDay=allday;
    }
    public boolean getAllday(){
```

```
        return allDay;
    }
    public CharSequence getLocation() {

        return location;
    }
    public CharSequence getTime() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

EventDataSQLHelper.java

The class is for creating the database that stores sights data.

```
package cityguide.informationRetrieval;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.provider.BaseColumns;
import android.util.Log;

public class EventDataSQLHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "sights.db";
    private static final int DATABASE_VERSION = 1;

    // Table name
    public static final String TABLE = "events";

    // Columns
    public static final String NAME = "name";
    public static final String LINK = "link";
    public static final String CLASSIFICATION = "classification";
    public static final String LATITUDE = "latitude";
    public static final String LONGITUDE = "longitude";

    public EventDataSQLHelper(Context context) {
```

```
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "create table " + TABLE + "( " +
            BaseColumns._ID+ " integer primary key autoincrement, "
+
            NAME+ " text not null, "+
            LINK+ " text not null, "+
            CLASSIFICATION+ " text not null, "+
            LATITUDE+" double, "+
            LONGITUDE + " double);";
        Log.d("EventsData", "onCreate: " + sql);
        db.execSQL(sql);
    }

    public void onClose(){

        super.close();

    }

}
```

SQLHandle.java

The class contains the method of getting sight data from the sights database and the method of adding sight data into the sights database

```
package cityguide.informationRetrieval;

import java.util.ArrayList;
import java.util.List;
import cityguide.map.Map;
import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
```

```
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.widget.TextView;

public class SQLHandle extends Activity {

    TextView output;
    Message message;
    EventDataSQLHelper eventsData;
    List<Message> messages;

    public Context context;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    public void onStop() {
        eventsData.onClose();
    }

    public void addEvent(Message message) {

        eventsData = new EventDataSQLHelper(context);

        SQLiteDatabase db = eventsData.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(EventDataSQLHelper.NAME,message.getName());
        values.put(EventDataSQLHelper.LINK,message.getLink().toString());

        values.put(EventDataSQLHelper.CLASSIFICATION,message.getClassfication(
        ));
        values.put(EventDataSQLHelper.LATITUDE,message.getLatitude());
        values.put(EventDataSQLHelper.LONGITUDE,message.getLongitude());
        db.insert(EventDataSQLHelper.TABLE, null, values);
        eventsData.close();
    }

    public List<Message> getEvents(){
```

```
        messages=new ArrayList<Message>();

        SQLiteDatabase db = eventsData.getReadableDatabase();
        Cursor cursor = db.query(EventDataSQLHelper.TABLE, null, null, null, null,
        null, null);
        startManagingCursor(cursor);

        while (cursor.moveToNext()) {

            Message message=new Message();
            message.setID(cursor.getInt(0));
            message.setName(cursor.getString(1));
            message.setLink(cursor.getString(2));
            message.setClassification(cursor.getString(3));
            message.setLatitude(cursor.getString(4));
            message.setLongitude(cursor.getString(5));
            messages.add(message);

        }

        eventsData.close();
        return messages;

    }

}
```

XmlPullFeedParser.java

The class is for parsing POIs&Events XML data

```
package cityguide.informationRetrieval;

import java.util.ArrayList;
import java.util.List;
import org.xmlpull.v1.XmlPullParser;
import android.util.Xml;

// parse the POIs&Events XML data
public class XmlPullFeedParser extends BaseFeedParser {

    public XmlPullFeedParser(String feedUrl) {
        super(feedUrl);
    }

    public List<Message> parse() {

        List<Message> messages = null;
        XmlPullParser parser = Xml.newPullParser();
        try {
            // auto-detect the encoding from the stream
            parser.setInput(this.getInputStream(), null);
            int eventType = parser.getEventType();
            Message currentMessage = null;
            boolean done = false;
            while (eventType != XmlPullParser.END_DOCUMENT && !done){
                String name = null;
                switch (eventType){
                    case XmlPullParser.START_DOCUMENT:
                        messages = new ArrayList<Message>();
                        break;
                    case XmlPullParser.START_TAG:
                        name = parser.getName();
                        System.out.println(name);
                        if (name.equalsIgnoreCase(SIGHT)){
                            currentMessage = new Message();
                        } else if (currentMessage != null){
                            if (name.equalsIgnoreCase(LINK)){
                                currentMessage.setLink(parser.nextText());
                            } else if
```

```
(name.equalsIgnoreCase(CLASSIFICATION)){

currentMessage.setClassification(parser.nextText());
    } else if (name.equalsIgnoreCase(LATITUDE)){

currentMessage.setLatitude(parser.nextText());
    } else if (name.equalsIgnoreCase(LONGITUDE)){

currentMessage.setLongitude(parser.nextText());
    } else if (name.equalsIgnoreCase(NAME)){
        currentMessage.setName(parser.nextText());
    }
    }
    break;
case XmlPullParser.END_TAG:
    name = parser.getName();
    if (name.equalsIgnoreCase(SIGHT) &&
currentMessage != null){
        messages.add(currentMessage);
    }
    else
        if(name.equalsIgnoreCase(RSS)){
            done=true;
        }
        break;
    }
    eventType = parser.next();
}
} catch (Exception e) {
    throw new RuntimeException(e);
}
return messages;
}}
```