Application development for the Android platform

Eniro Friend Finder

Axel Johansson
Jakob Sahlström
Abstract

Application development for the Android platform

Axel Johansson, Jakob Sahlström

This thesis is an idea that derives from Eniro AB. The idea was to make an application for mobile devices that would make it easier for the user to stay in touch with friends. The application should be able to gather information of the phone’s contacts by searching Eniro for their phone number and present the information in a simple and convenient way. Because of the authors' previous experience in Java programming the application was made for Android devices. Android provides a software development kit with necessary tools and documentation for developing applications with the Java programming language.

The result is an application that is able to search Eniro for all contacts in the phone book and store them in a database. These are then presented in both a list view and a map view. The list is sorted with the person living closest to the user’s location at the top. From the list view the user has the option to either call the contact or send a text message. The map view contains a map with markers at the contacts’ locations. By clicking the marker certain information will be shown in a window.

There are still some bugs to fix and features to implement. Possible improvements to the application could be functions for finding restaurants and cafes nearby and a less time consuming synchronization. Since the project was to be done within a limited period of time and the authors did not have any previous experience of Android development the application should be considered as a prototype.
7.2 Problems encountered ........................................... 26
7.3 Bugs ........................................................................ 27
7.4 Possible improvements .............................................. 28
7.5 HCI perspective ......................................................... 28
7.6 Lessons learned ......................................................... 29

A Time plan ................................................................. 32
1 Introduction

Since the launch of Android in late 2007 [1] the occurrence of mobile devices has increased heavily. Roughly 9 billion applications have been downloaded from Android Market and the number is steadily increasing.[2] The cell phones of today are becoming more and more like small computers that people carry around wherever they go. As people have become aware of the fact that information nowadays is very easily accessible the need for information has increased. For instance, it is no longer hard to find out where people live. The days of the physical Yellow Pages are over!

1.1 Background to the project

Students that study in other places than their hometowns like a reminder application that reminds them about their old friends once back home for Christmas or over the weekend. When using the application the user will be able to see which of his or her contacts live nearby and who is closest to the user’s location. With both a list view and a map view your old friends from school are just one click away!

The mission is to build an Android application, called Friend Finder, that takes the contacts in the cell phone and uses Eniro’s search engine to provide GPS coordinates for where the user’s friends live. Then by calculating the distance between the user’s location and the user’s contacts’ GPS coordinates a sorted list of the contacts will be shown with the contact with the shortest distance on top. The original case description can be seen in Figure 1.
Friend reminder

Background
Students that study in other places than their hometowns like a reminder application that reminds them about their old friends once back home for X-mas or over the weekend.

US0 (Theme: UI design and interaction)
Based on the stories US1 – US6 design and document an in app map. Advice less is more and simplicity over complexity.

US1 (Theme: Android Contact API integration)
App should once initiated connect with the contact information on the devices (or dialog) and retrieve the phone numbers of those contacts.

US2 (Theme: Eniro API integration)
App should perform searches on the phone numbers from US1 towards the api.eniro.com person search services. If a hit is returned store the contact info, name, number and coordinates in memory.

US3 (Theme: Android Maps API integration)
App should retrieve the GPS location of the devices.

US4 (Theme: Android Maps API integration)
App should compare the distance between device and hits stored from US2 and sort them by distance.

US5 (Theme: Android Maps API integration)
App should present US4 in a list (sorted by distance), the user can now switch to map mode and all hits including the device itself are presented on a map.

US6 (Theme: Android Maps API and Eniro API integration)
In map view or list view (design yourself) it should be possible to suggest nearby cafés, restaurants or pubs. (Includes integration with api.eniro.com company services).

Themes in order of appearance
- UI design and interaction flow “app map”.
- Android Contact API integration (can be replaced by just simple enter number dialog)
- Eniro API Integration (person and companies services)
- Android Maps API Integration (preferable google maps given the scope, includes GPS).

Figure 1: Original case description
1.2 About the course

The project has been done within the scope of the course Independent Project in Sociotechnical Systems Engineering - IT Systems (15 credits) that culminates in a Bachelor’s degree from Uppsala University.

1.3 Previous knowledge

Axel Johansson (student at Master Programme in Sociotechnical Systems Engineering) has had use of the courses Algorithms and Data Structures I (5 credits), Database Design I (5 credits), Human Computer Interaction (5 credits) and Programming (10 credits), all given at Uppsala University. Jakob Sahlström (student at Master Programme in Engineering Physics) has had use of the courses Computer Programming (5 credits) and Object-Oriented Programming (10 credits), all given at Uppsala University. Neither Axel or Jakob had any experience of Android development when going into this project, their programming experience extended to Java only in Jakob’s case and Java and Python in Axel’s case.

2 Requirement and specification

The objective of this project is to get introduced to “real life” problem solving in the field of software development and to learn more about different types of methods used in this field, such as pair programming and collaborative programming. The objective also contains introduction to different kinds of languages needed in software development with Android.

The objective for Eniro is to test a pilot project and find out whether or not the original idea Eniro had is worth continue working with. Even though the authors are concerned with the completion of a usable product the objective is more than that. Since this is the first time the authors will work in a project like this the objective is also to gain experience from working in a software development project. Thus, the process of learning, working according to a time plan, working in a group and finishing the product are all of equal importance.
3 Tools used for developing the application

The tools used in this application are the Android platform for mobile devices and Eniro’s different services. Android contains an operating system, middleware and key applications developed by Google and is used in phones, tablets and other portable devices.[3] Android’s main competitor on the market today is Apple’s operating system, iOS, which is used in e.g. iPhones and iPads.

3.1 Android architecture

The Android platform runs on a Linux kernel and contains several libraries and application frameworks. To start developing Android applications the Android SDK (Software Development Kit) is needed. The Android SDK provides necessary tools and API:s (Application Programming Interface) for developing your own applications with the Java programming language. See Figure 2 for the full platform architecture.[3]

An API is a documentation or a specification in form of source-code intended as an interface in software development. When using an API all software components within the API can communicate with each other. An API may include several specifications in how to use the components, e.g. data structures, routines, object classes, methods and variables.[4] In this project API:s will mainly be provided by Eniro and Android SDK.
3.2 Main components

3.2.1 Folder architecture

When creating a new Android project in an integrated development environment such as Eclipse, a certain folder architecture is automatically generated. The Android SDK helps the developer to structure his or her work from the very beginning. The folders mainly used are:

- /src - Contains all the source code in form of .java files
- /gen - Contains an auto generated .java file, R.java, which job is to gather all resources in form of integer constants. These resources are defined in the project’s .xml files. R.java is updated every time the project compiles and therefore it should never be configured manually.
- /assets - Contains raw assets files such as .html files.
- /res - Contains all the resources of the project, such as layout files, pictures, shapes, strings and colors.
• AndroidManifest.xml - Presents essential information about the application, see section 3.2.4.
• Referenced Libraries - External libraries used in the application that is not included in the Android SDK by default.[5]

3.2.2 Activities

An activity represents the UI (User Interface). In other words, the activity is what is shown on the screen and the user may interact with. An activity can cover the whole screen or just a part of it. An application is usually made up by several activities, depending on the complexity of the application. Since an application uses several activities, it is important to know about the life cycle of an activity. All activities extend the class `Activity` and it contains especially 6 important methods that describe the state of the activity.[6] See Figure 3.

When an activity first created the method `onCreate` is called in which the layout and content of the GUI is set up. Then the methods `onStart` and `onResume` is called and the activity is now running. The activity can then start new activities by a messenger called `intent`. An intent is created with an `Intent` object, which defines a message to activate a new activity. When starting a new activity, the first one calls `onPause`. If the new activity covers the whole screen `onStop` is called. When going back from the new activity `onResume` or `onStart` is called depending on if the old activity was paused or stopped. When the user exits the application `onDestroy` is called.[6]
Figure 3: The Android activity life cycle illustrating the different states of an activity.

3.2.3 XML Layout

Android offers a convenient way of setting the layout of an application. The UI can be setup in the code and/or by declaring the UI elements in XML. This is
a smart way of separating the layout design from the code that handles the user interactions. The developer may also use different XML layouts depending on different circumstances, e.g. different screen sizes or platform version. Listing 1 shows an example of .xml code.[7]

Listing 1: .xml code example

```xml
<?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>
```

A general problem with Android development is that Android devices exist in all kinds of screen sizes, resolutions and platform versions. This may cause issues regarding layout design and available components, e.g. a big button may look good on a big screen but terrible on a small one. The Android SDK provides different methods to deal with these kind of problem, one is the possibility to use Density-independent Pixels (dp) which is a virtual pixel used when defining layout dimension in a density independent way.[8]

### 3.2.4 The AndroidManifest.xml File

The Android Manifest file is an .xml file presenting essential information about the application to the Android system. This information is needed before the system can run any of the application’s code. The manifest describes the components of the application and declares permissions needed in order to access protected parts
of the API, e.g. permission to access internet or GPS tracker. It also determines which processes will host the application components.[9]

### 3.3 Eniro’s API

The authors used two of Eniro’s API:s from api.eniro.com, person search and Eniro maps. These API:s are based on JavaScript and HTML.

#### 3.3.1 Person search

A person search is made by configuring an URL with, in this case, a phone number. The response is a string in form of a JSON (JavaScript Object Notation) object that contains contact information about this phone number. By using the Gson library, provided by Google, certain information such as name, street name, post area and coordinates is extracted to a comfortable format used in Java. The Gson library provides methods to convert JSON objects to Java objects.

#### 3.3.2 Eniro maps

Eniro provides lots of examples of how to use different objects from the Map API. By having a HTML file in the android project and a JavaScript bridge between Java and Eniro’s API the user is able to configure the map interface in several ways. The HTML file contains modified versions of the examples provided by Eniro.
4  Theory

Since the application should be able to sort the cell phone’s contacts by distance from the user the application needs to calculate the distance between two coordinates and then sort the contacts.

4.1  Haversine formula

To calculate the distance the haversine formula (1) is used. The formula uses spherical trigonometry to calculate the distance between two points on the Earth’s surface specified in longitude and latitude, assuming the Earth is a sphere. The distance can be calculated as

\[ d = 2r \arcsin \sqrt{\sin^2 \left( \frac{\phi_2 - \phi_1}{2} \right) + \cos \phi_1 \cos \phi_2 \sin^2 \left( \frac{\psi_2 - \psi_1}{2} \right)} \]  

(1)

where \( d \) is the distance [m] between two points with longitude and latitude \( (\psi, \phi) \) and \( r \) is the mean Earth radius [6,371,000m].[10]

4.2  Heapsort

Heapsort is a recursive sorting algorithm with a runtime upper bound of \( O(n \log(n)) \). The algorithm is built on two steps. The first step is to build a min-heap out of the data array where the indices of the elements in the array are ordered from the root and from left to right, see Figure 4. The second step is to remove the smallest element from the heap and put it in the sorted array. By repeating step one and two until all the elements are removed from the unsorted array the sorted array is built.[11]
Figure 4: The structure of a min-heap
5 Method

5.1 Programming method and environment

When developing the application the methods used have been collaborative programming and pair programming. Pair programming consists of two persons where one is the “Driver” who writes the code and the other is the “Observer” who makes sure the code is correct. The roles are switched continuously.[12] The integrated development environment has been Eclipse and the plug-in Saros has been used for the collaborative programming. Saros gives the developers the possibility to edit the same code simultaneously. At the start an Android emulator plug-in was used to test the code. After a while the authors realised that it was possible to test and run the code directly in the cell phone using HTC Sync. Since the application needed access to the cell phone’s contact list this proved to be a much better way to test the application. Also, it was considerably quicker than the emulator. An extra bonus to the developing process was the fact that the authors had devices with different versions of the Android platform which proved useful when testing since several bugs only appeared in one of the versions.

The coding has been planned and written step-by-step and emphasis has been put on coding and testing every method separately to make sure that it works properly. Every week has had coding goals that were to be reached, see Appendix A, and feedback has continuously been given to the time plan to make sure that the work progressed according to it. In the later phases of the project the application has been tested in realistic situations, for instance walking around downtown and traveling by train.

The code has been written to support version 2.2 or higher of the Android platform which 93.5% of all users of Android devices use.[13] The reason for not making the application applicable for even older versions was that several functions that has been used in the application are not supported by older versions of the Android platform. Since the application will be forward compatible, i.e. applicable to newer versions as well, and the older versions become increasingly
rare, this proved to be the best solution. Figure 5 shows the workspace used when programming.

![Workspace Image]

**Figure 5: The workspace used**

1. The Package Explorer provides an overview of the project and all files that it contains including .java files, images, R.java, .xml files and other assets and resources.
2. The Editor where the code is edited.
3. The Console displays the progression of the install and eventual installation errors.
4. The Saros tab displays the programmers that works in the same session. There is also a chat window the programmers can communicate through.
5. The logCat displays runtime errors, i.e. errors that occur while running the application. By hovering an error more information is displayed in a dialog.

### 5.2 Gathering of information

Since the project idea derived from Eniro some of the work has been done at Eniro’s office in Frösunda, Stockholm. At the office the authors had the possibility to ask for technical advice from several Eniro employees as the project progressed.
The help from employees has concerned extraction of data from Eniro’s API:s as well as coding issues in general. Also, working in a real-life development environment gave the authors experience of how the daily work in a company works.

Even though the employees at Eniro have contributed with valuable advices most of the information has been found by searches on Google. Every so often the hits have led to developer.android.com and www.stackoverflow.com where someone else has posted the problem before. Since the application do not have remarkably advanced functions the solutions to the problems that the authors have faced have been fairly easy to find. The credibility of the sources of information have been very easy to test, if the code does what it should it is obviously a trustworthy source.

5.3 The group work in practice

This project has been done during a period of ten weeks. The work has mostly been done in the project room provided by Uppsala University where other project groups also have been working. Because of the distance to Eniro’s office in Frösunda, Stockholm the time spent working there has been limited due to economic- and time-related reasons. The authors have spent approximately one day a week working at Eniro’s office.

The work has largely been characterized by cooperation. Since the authors have been working beside each other the communication within the group has been very quick and direct. Each one in the group has been responsible for different parts of the code. One has for instance been responsible for the database while the other one has focused more on the layout. Despite this split the authors have asked each other for help when needed and all the code has been reviewed by both of them. The report has been written in the same manner.
6 Design and implementation of Friend Finder

The final result is an Android application that conforms to the original idea. It contains two activities with different functions, one handles the synchronization with the cell phone’s contacts and Eniro’s API and the other one handles the presentation of the contacts that live nearby the user. Below follows database design, print screens, explanatory text and pseudo code.

6.1 Database design

In order to store the contacts’ address info a database was designed and implemented in the application. Retrieving all the data from Eniro’s API is a quite time and bandwidth consuming process and it would be very inconvenient to retrieve all the data every time the application is started. Thus, a database class that extends the SQLiteHelper class was designed. SQLiteHelper is a class that is used to implement databases in Android. The database class (ContactDatabase) mainly consists of one large table where each row is filled with information about one contact such as contact name, name stored in Eniro’s API, phone number, address and coordinates. Every time the application starts this information is fetched from the database and stored in an ArrayList that is used while the application is running. The other table that is stored in the database is very simple, it contains one single value, namely the max radius for viewing contacts that the user has chosen. In order to maintain this radius when the application is stopped it was necessary to store this value in the database and since it is not related to a certain contact it was put in a separate table. Figure 6 illustrates the database in an ER diagram.
6.2 Application design and functionality

Figure 7 is the first screen that is displayed to the user. When pressing “Visa vänner” the application checks if there are any contacts stored in the database. If
not, a dialog pops up that asks if the user wants to synchronize. Before the synchronization is executed the application makes sure there is an internet connection otherwise it is not executed. In the menu that is also displayed in this picture it is possible to synchronize manually, to read about the application and to exit it. The pseudo code for this activity can be seen in Listing 2.

```
public class FriendFinderActivity extends Activity{
    public void onCreate(Bundle savedInstanceState){
        //Sets the layout.
    }
    public void onStart(){
        //Connects to the database to retrieve data from the contact table.
    }
    ...
    //Methods for synchronizing, checking the internet connection and connecting to the database.
    ...
}
```
Figure 8 shows the user interface while synchronizing the user’s contacts with Eniro’s API. When synchronizing it is not possible to do anything but wait. A progress dialog displays the progress. The contacts that does not have address data including coordinates on Eniro are ignored and not added in the database. The pseudo code for how the information is retrieved and stored is shown in Listing 3.

Listing 3: Back-end - Synchronization

```java
public class Contact {
    public Contact(String phoneNumber, String thePhoneName, Context context) throws IOException {
        //Constructor used to create Contact objects by searching on Eniro's API and then store the data in instance variables such as phone book name, street address, coordinates etc.
    }

    public Contact(String[] personData, double[] coordinates) {
        //Constructor used to store data taken from the database in the Contact object. Thus, when the
```
contact already is in the database there is no need for searching for it on Eniro's API.

... //Get methods for the variables such as name, phone number, address and the distance to the user. //Methods for retrieving data from the JSON objects that are returned from Eniro's API.

...)

class ContactDatabase extends SQLiteHelper{
    public void onCreate(SQLiteDatabase db){
        //Creates the tables and views needed in the database.
    }
}

...//Methods for adding contacts, setting radius and updating values.

...

(a) List view          (b) Radius setting dialog          (c) Map view

Figure 9: User interface after synchronization
Figure 9a displays all the contacts ordered by the distance to the user with the contact closest to the user first. The distances are calculated with Equation (1) and the list is sorted with the Heapsort algorithm, see Section 4.2. By clicking a contact a new list pops up under it where the user can choose to either call or send a message to the contact. In the menu that is also displayed in this picture it is possible to view settings, to re-sort the list, update the map and to read about the application. The list is also re-sorted when the user changes from the map view to the list view. When the user changes back and forth between the views the map is not reloaded in order to minimize bandwidth consumption. In Figure 9b the user can choose to view contacts that live within a specified radius from the user’s current location. The contacts that live within this radius will be displayed in the map view. Figure 9c displays a map where the user’s position is marked in red and the contacts that live nearby are marked in yellow. By clicking a contact an info window pops up that declares which contact it is. The map is zoomed automatically to show all the contacts that live within the chosen radius. The user can zoom manually and also choose different views such as aerial photo view. Listing 4 shows the back-end of the user interface after synchronization.

Listing 4: Back-end - User interface after synchronization

```java
public class FFTabWidget extends TabActivity implements OnTabChangeListener {
    // The class where most of the work is done. A TabHost manages the list and map tabs. Since the list view and map view are handled and designed in this class the back-end perspective of these two have been merged.
    public void onCreate(Bundle savedInstanceState) {
        // Sets the layout.
        // Creates the list and map view.
        // Creates the JavaScript bridge needed to interact with the Eniro map API in the .html file.
        // Starts searching for the user location.
        // Retrieves the Contact list from FriendFinderActivity, calculates the distances
```
\ with the Haversine equation and uses Heapsort to sort the list according to the distances to the user.

```java
public void onStart(){
    // Connects to the database to retrieve data from the radius table.
}
...
```

// Methods for updating the distances to the user, the set radius dialog, searching for the user's location, setting up the tabs, create and update the list view and map view, connecting to the database.
...
```
7 Discussion

7.1 Decisions made

The decision to use the heapsort algorithm was made because of the low upper bound limit and the property that the smallest element is extracted first. Thus, heapsort gives the possibility to stop sorting when a certain distance is reached. If the user has many contacts, this property could decrease the time needed when sorting. When realizing that the sorting was executed very quickly this feature was not implemented. The Haversine formula is a way of calculating distances between coordinates that functions well because the contacts registered at Eniro are located within a limited area (Sweden). The assumption that the Earth is a sphere is therefore negligible. Finally, the choice to use the Gson library was made because the API was easy to understand and implement.

7.2 Problems encountered

Since the person search API sends back a JSON object there was some trouble extracting that information to a convenient format in Java. The final choice was to use the Gson library provided by Google that contains methods to convert a JSON object to a Java object. A discovery was made that the order of imports mattered and the Gson library had to be imported first for the application to work.

A lot of time was used to understand and debug the components used to retrieve location updates. It is possible to get location updates from three different providers; namely GPS, Wi-Fi or 3G. The application is supposed to take the location update from the provider with the best accuracy. The GPS generally provides the most accurate location but only works with a clear sky. Therefore, a problem occurs when inside; the GPS gives the last known location which is the most accurate, but it’s not the right location. By adding a time condition where the last location had to be within a 30 seconds interval from the runtime the problem was solved.

There were some problems regarding how to get in touch with the user’s contacts via the application. The best would be to let the user choose between all
available ways of communicating, i.e. phone call, text message or email. This could be done with a class called QuickContactBadge but after struggling with the class documentation and realizing the time was too short a decision was made to use an ExpandableListView instead with just the options to make a phone call or send a text message.

By using the Eclipse plug-in Saros the coding was expected to go faster because of the possibility to write at two places at the same time in the same code. Unfortunately Saros often had problems synchronizing the code between the computers which caused a lot of distractions in the workflow. This may have been a consequence of UpUnet-S being an internet connection with unreliable bandwidth.

With HTC Sync the testing of the application could be done directly on the cell phones. Sometimes, when re-installing, the console gave an error message saying 'Different application signatures’. By uninstalling the application and then installing it again that error disappeared. For some unknown reason the phone with the oldest Android version sometimes got an error, when trying to install the application on the internal storage, telling 'Insufficient storage’ despite the fact that the installation went perfectly just minutes before. When moving the installation to the external storage a new error message telling 'Container error’ showed up. By reconnecting the device and re-install the application the error message finally disappeared. These problems have led to several moments of frustration.

7.3 Bugs

- For unknown reasons some phones may have a hard time clicking the markers in the map view. Not sure if it depends on android platform version, screen resolution or Eniro’s Map API.
- When having all contacts shown in map the word 'Alla’ is not shown in the settings dialog when reopening the dialog.
- When having a radius with a non-zero decimal the number is floored next time the settings dialog is opened. The user might think the radius is changed but that is not the case.
7.4 Possible improvements

- More ways to communicate depending on what information is stored in the phone contact (mail, Skype etc.)
- When synchronizing, only synchronize those contacts that are not yet in the database for a faster user experience.
- It should be possible to run the synchronization in the background, for instance by pressing the home button - the application should not stop by that.
- Different Android themes for different platform versions.
- Retrieving old versions of the database.
- Non-linear (perhaps logarithmic) scale when setting view radius since it is tricky to set a small radius value. Other solutions would be plus/minus buttons at the side for step way adjustment or a wheel to pick radius.
- Provide the opportunity to show the N closest friends.
- Zoom adjustment only run when needed.
- Filter by group (Family, friends, colleagues)
- Facebook integration.
- Display restaurants, cafés, cinemas, theaters nearby.
- Numbered markers on the map that corresponds to the contacts in the list.
- It should only be possible to set the radius in the map view since it is there the radius is used. Maybe the list view and the map view should be in separate activities.
- A database that handle several addresses and is centered around the phone number instead of the name.
- Have the database handle the sorting of the contacts.

7.5 HCI perspective

While developing the application the Human Computer Interaction (HCI) perspective has always been kept in mind. Eniro’s original specification explicitly said “less is more and simplicity over complexity”. The original case description is included in Appendix 1. The authors’ goal has been to work according to these
advices by making the application as intuitive and easy to use as possible, for instance by adding few buttons and require few clicks. The colour scheme has been chosen both to be pleasant to look at and match the color schemes of Eniro. By adding options and alternatives in the menu the user is allowed to modify the application manually without disturbing the interface. This provides a clean and simple interface that uses the whole screen.

7.6 Lessons learned

Developing an Android application for the first time has been a very educative and challenging process and there are some things that the authors would have done differently if they were to redo the project. It would have been less time consuming to create several small applications to more easily and safely test the different parts of the code. Sometimes perfectly functional code may be destroyed by adding new features and thus create bugs instead. This way of testing was implemented when creating the database but not for the location search which would have been a good idea since it is the code that has been the most troublesome. This lesson is highly related to the authors’ lack of experience and theoretical knowledge as the project began. Probably, it would have saved some time to dig deeper into the theory about fundamental components of Android development before starting to write the code.

Another thing that the authors should have realised earlier was the fact that Saros did not work properly. Therefore it would have been better to stop using it and switch to Apache Subversion, an open-source version control system used and recommended by Eniro. By using Subversion, the developers can upload the code to a server and synchronize their code through that instead of sharing peer-to-peer.

Finally, the authors have found it very interesting and fun to develop an Android application. It has become apparent how much one can learn by doing an independent project like this but also how much there is yet to discover. The authors hope to get the opportunity to develop more applications for Android and to continue to discover the immense world of knowledge that is programming.
References


A  Time plan

Table 1: Time plan for Friend Finder

<table>
<thead>
<tr>
<th>Week</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Write project plan and download necessary tools/software for programming.</td>
</tr>
<tr>
<td>13</td>
<td>Become familiar with Eniro:s API:s and write a simple java program that uses this information.</td>
</tr>
<tr>
<td>14</td>
<td>Start Android programming. Set the application layout</td>
</tr>
<tr>
<td>15</td>
<td>Write a small Android application for retrieving personal contact information and synchronise it with the information stored on Eniro’s API:s.</td>
</tr>
<tr>
<td>16</td>
<td>Write code for information storage, including synchronisation between the contacts in the phonebook and the information on Eniro’s API:s.</td>
</tr>
<tr>
<td>17</td>
<td>Prepare and present at midterm presentation</td>
</tr>
<tr>
<td>18</td>
<td>Write code for presenting the contacts locations on a map and general design.</td>
</tr>
<tr>
<td>19</td>
<td>Polish the code and test the application.</td>
</tr>
<tr>
<td>20</td>
<td>Write report.</td>
</tr>
<tr>
<td>21</td>
<td>Deadline - report. Prepare opposition</td>
</tr>
<tr>
<td>22</td>
<td>Prepare presentation</td>
</tr>
</tbody>
</table>