

GPS Tracking System

- Plotting and Storing Maps using the Web and the Cell Phone



**LUNDS
UNIVERSITET**

Lunds Tekniska Högskola

**LTH School of Engineering at Campus Helsingborg
Computer Science**

Bachelor thesis:
Ehsan Neamat

© Copyright Ehsan Neamat

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2008

Abstract

The bachelor thesis was to create a GPS tracking system for Svep Design Center AB. The system would consist of two major parts which were linked together through a common database. One part consisted of a cell phone application where the user can transmit their current position and the other part was a web application where the position data is presented on a map outlining the route of the user.

Since such a system is rather big the goal was to implement a base that later could be marketed and grow into a full scale commercial product.

Difficulties with the project including designing a database being able to handle great amount of data while being optimized with speed and performance. Also the application for the cell phone needed to be very user friendly, use minimal resources and have a low data traffic to keep the costs down.

An iterative project model was used when implementing. First a database was created, then the main structure of the cell phone and web applications were coded and finally extra functionality was developed and added.

When the main functionality had been implemented the system was tested. A route of approximately 49 kilometres with a duration of 50 minutes was tracked while driving. GPS data was sent periodically with an interval of 30 seconds. The tracking had consumed 8% of the cell phone battery. 77 coordinates were sent to the database corresponding to 29kb of data. The total cost was 0.77 Swedish kronor.

To sum it up the goal was achieved and a stable base of a GPS system with social networking features was developed which can be marketed and built on commercially.

Keywords: GPS, tracking system, .NET C#, Java, J2ME, MSSQL, Google Maps

Sammanfattning

Examensarbetet gick ut på att skapa ett GPS spårningssystem för Svep Design Center AB. Systemet skulle bestå av två huvuddelar som var sammanlänkade med en gemensam databas. Ena delen bestod av en applikation för mobiltelefonen där användaren kan skicka sin nuvarande position och den andra delen var en webbapplikation där positionsdatan presenterades på en karta och ritade upp användarens rutt.

Då ett sådant system är ganska omfattande var målet att implementera en fungerande bas som senare kunde marknadsföras och växa till en fullskalig kommersiell produkt.

Svårigheterna med projektet var bl.a. att designa en databas som klarar av att hantera stora mängder data samtidigt som den är optimerad för hastighet. Vidare var det vitalt att applikationen för mobiltelefonen var användarvänlig, utnyttjade minimala resurser och höll en låg data trafik för att minska kostnaderna.

En iterativ projektmodell användes vid implementering. Först skapades databasen, sedan kodades huvudstrukturen applikationerna för mobiltelefonen och webben och slutligen utvecklades extra funktionalitet som byggdes på.

När huvudfunktionaliteten var implementerad testades systemet. En rutt på ca 49 kilometer som varade i 50 minuter kartlades under körning. GPS data skickades periodvis med ett intervall på 30 sekunder. Kartläggningen konsumerade 8% av mobiltelefonens batteri. 77 koordinater skickades till databasen vilket motsvarade 29kb data. Den totala kostnaden blev 0.77 svenska kronor.

För att sammanfatta det hela så uppnåddes målet och grunden för ett stabilt GPS system med sociala nätverksegenskaper upprättades som kan marknadsföras och vidareutvecklas kommersiellt.

Nyckelord: GPS, tracking system, .NET C#, Java, J2ME, MSSQL, Google Maps

Foreword

This project was born from a series of brainstorming sessions at Svep's coffee table during the morning meetings. It was aspired to create a community like application where users could share their routes and journeys with each other, not only in text or photo form but rather in a combination of these along with a map illustrating the route. Such an application would take the online socializing experience to a whole new level and would be of great benefit for both the user himself as well as those whom he associates with.

There were a few similar existing applications however they had not managed to establish themselves on the internet due to a couple of reasons. The primary reason was that they lacked a proper marketing and had shut in their application behind closed doors which didn't really attract a new visitor to sign up and look inside. Also some of them were targeted to certain groups (for example customers of a certain phone model or brand) and hence other user groups were left out.

However Svep's initiative was truly innovative in the sense that it was targeted towards the masses and restricted to Swedish users. The users who wished to share their maps only needed a mobile telephone and a GPS device of free brand and those who wanted to check out others' maps only needed to sign up on the website. Since all users are based in Sweden it would generate a closer link between users and to real life. When it comes to marketing it was important that high prices and membership fees does not scare of potential users. Rather the approach was to open up the application inviting users to try it out and within time, when the popularity of the application increased, present a neat charging method where Svep would profit from the application without losing users.

Svep presented the idea to me in early November 2007 and a short time thereafter I started the project. The project has been of great benefit for me in terms of knowledge, understanding, planning, future insight, responsibility and of course software development and many lessons have been learned.

I would like to sincerely thank all involved parts at Svep Design Center, especially Lars Gustavsson who believed in me and took me onboard on this project, Bo Nyman who guided me along the way towards successful end prototype and all the developers who helped me during the process. I would also like to thank my mentor Mats Lilja for all assistance in making this possible.

Helsingborg (Sweden), May 2008

Ehsan Neamat

List of contents

1	Introduction	1
1.1	Background	1
1.2	Objective	1
1.3	Purpose	2
1.4	Problem description	2
1.5	Delimitation	3
2	Methodology	4
2.1	Project model	5
2.2	Research on the system and all relevant aspects	5
2.2.1	The construction of the GPS device (HGE-100)	6
2.2.2	J2ME	6
2.2.3	.NET framework	6
2.3	Designing the database	6
2.4	Constructing the main structure of the MIDlet	6
2.5	Establishing a network connection between the MIDlet and the server	6
2.6	Constructing the main structure of the website	7
2.7	Retrieving GPS data from the GPS device	7
2.8	Transmitting the GPS data	7
2.9	Presenting the GPS data on the website	7
2.10	Adding functionality to the MIDlet	8
2.11	Adding functionality to the website	8
2.12	Adding support for built in GPS	8
2.13	Marketing the system	8
3	Implementation	9
3.1	Research on the system and all relevant aspects	9
3.1.1	The construction of the GPS device (HGE-100)	9
3.1.2	J2ME	9
3.1.3	.NET framework	10
3.2	Solution overview	10
3.3	Designing the database	11
3.4	Constructing the main structure of the MIDlet	12
3.4.1	How a MIDlet works	12
3.4.2	J2ME User Interface Architecture	13
3.4.3	Paper prototypes of the different MIDlet screens	14
3.4.4	Implementing the first screens	16
3.5	Establishing a network connection between the MIDlet and the server	16
3.6	Constructing the main structure of the website	18

3.6.1 The login module	20
3.7 Retrieving GPS data from the GPS device	21
3.7.1 Establishing a connection	21
3.7.2 Controlling the data flow	21
3.7.3 Parsing the NMEA data	22
3.7.3.1 Analyzing the NMEA data	22
3.7.3.2 Treating the NMEA data	23
3.7.4 Integration into the MIDlet	23
3.8 Transmitting the GPS data	23
3.9 Presenting the GPS data on the website.....	24
3.10 Adding functionality to the MIDlet	26
3.10.1 List and create routes	26
3.10.2 Send messages.....	26
3.10.3 Take photos	27
3.10.3.1 File attachment.....	28
3.10.3.2 Photo capture.....	30
3.10.4 Offline mode.....	31
3.11 Adding functionality to the website	32
3.11.1 Social connections.....	32
3.11.1.1 Friends.....	33
3.11.1.2 Groups	33
3.11.2 Dashboard	33
3.12 Adding support for built in GPS.....	34
3.13 Marketing the system.....	36
3.13.1 Logistic.....	36
3.13.2 Sport.....	36
3.13.3 Travelling	37
3.13.4 Community	37
4 Description of the system	40
4.1 The MIDlet	40
4.2 The website	41
5 Result.....	43
6 Summary	45
6.1 Difficulties	45
6.2 Improvements	45
6.3 Lessons learnt.....	46
6.4 Last words	46
7 References.....	48

1 Introduction

1.1 Background

In November 2007 the Swedish engineering company Svep Design Center presented an idea for me. They had previously created a GPS device for mobile phones and now they wished to develop an application where users could keep track on each other. One of the user scenarios which were put forth was the Vasa ski race. The thought was that if a person races in the Vasa ski race the rest of the family and friends should be able to keep track of the person on a map: what his route is and where he is located. He should also be able to send messages to tell his viewers what he is currently doing, like for example: *"I am at a lake drinking blueberry soup!"*

When I grasped the idea and ponder upon it for a while I realized it could be a real success if developed and marketed correctly. Apparently very few had thought of this idea before, to store coordinates and present them on a map, in a wider scale. The applications which already existed were limited and not very attractive for use on a broader scale.

I was told in the meeting that they were thinking something in the lines of a cell phone application using GPS to send coordinates, a database storing the coordinates and a Windows application which presented the coordinates in the form of a route on a map. We discussed possible realizations of the idea and I suggested that instead of a Windows application a web site should be created where anyone could register and watch routes which they were authorized to. Since a web site is platform independent it enables users from all platforms and operative systems to use the application. Apart from this no installation is needed and users can access their routes and watch the routes of others wherever they are, as long as there is an internet connection.

This suggestion was appreciated and it was decided to start implementing the idea. What was once an innovative thought had now become a highly promising project.

1.2 Objective

The objective of the project is to create the base of a system which stores and exchanges routes based upon GPS data. The system will consist of two main parts: a MIDlet and a website. By MIDlet an application for the cell phone is meant which is used to send GPS data for storage. By website a web application for the computer is meant where the data is presented and exchanged with other users. A database will connect the two where all data is stored and fetched.

Along with the system a short study will be made on what areas of use there are for the system to be applied to.

1.3 Purpose

The purpose is to create a system which enables end users to share their routes and journeys with each other. The aim is to take social interaction into a higher level where ordinary text and multimedia is combined with visual maps of the user's personal routes and journeys. Such a system will have many areas of use: from social online communities to tracking systems for companies, which the study later will demonstrate.

1.4 Problem description

So far there has been blogs, training journals, travelling diaries, photo communities and the like. GPS and map applications also exist but there are no proper systems which combine all of these features. Merging all these into one creates a broad system allowing the users to track each other in real time, visualise their routes on a map, storing their routes and let the user share it with others. Such system takes socializing into a whole new dimension and have many areas of use but will also face many problems and obstacles on the way to realization.

These difficulties occur since the system will have many subdivisions in order to function as desired. Considering how huge the database will grow due to all coordinates being sent by users it is vital that the core of the database is designed in such a way that it is optimized in speed and performance. Additionally it must be able to meet future demands and new features without major modifications.

The dynamic structure allowing easy adaption to new features is also an important factor when it comes to the website. Even though the focus is on visualizing user routes on maps, and hence this section should be coded for optimal performance and minimum computer demands, there are a great range of add-ons which could easily be integrated into the website. Also if the core is coded cleverly it will be fairly effortless to customize the website for any area of use.

As far as the MIDlet is concerned the main goals should be user friendliness and minimal resource usage. Considerable aspects for achieving high user friendliness are clear and understandable menu structures and removal of unnecessary elements. The key word for a user friendly MIDlet is simplicity in content, design and features. Further minimal resource usage consists of several factors, all of significant importance when it comes to cell phones. These primarily involve low battery usage and low data transmissions (to minimize the cost).

1.5 Delimitation

Due to the comprehensive nature of the project only the core element needed for the system to function properly will be implemented. These consist of a database for storage, a website for visualization of user routes and a MIDlet for sending coordinates. Once the elementary parts are implemented and interact relatively smoothly the practical fraction is considered to be done. As a complement a small theoretical study will be made where possible areas of use are suggested to give a hint of what the system could be used for.

Since the graphical interface and design of the website and MIDlet depend upon what the system is used and what group of users are targeted it will be left rather simple. Instead the focus will be on the functionality under the shell.

2 Methodology

The approach to realize the project will be as follows: firstly there will be a research period on the system itself to get a clearer picture of what it is to be used for, who it is targeted to, what it is expected of it and how the future might look. There will also be research on how the previously created GPS device by Svep is coded and works, along with how it is possible to extract the GPS data and send it away for storage in a database. Due to cell phones mainly supporting Java in regards to applications coded for the cell phone, a closer look at J2ME and the structure of a MIDlet will be taken. Since Svep has a Microsoft server environment it was desired that the website and database to be implemented in the Microsoft developing tools .NET and SQL. Hence a look at the .NET framework and an understanding of its many components will be included in the research.

When the research has been made the database will be designed. Since the research will have brought down the project from the abstract level of ideas to an understandable level it will be easier to know what is demanded from the database. Consequently the database can be coded in such a way so future changes hopefully would not affect the database structure to much.

Thereafter the main structure of the MIDlet will be designed. Efforts will be made to establish a network connection between the MIDlet, the server and the database. Once the connection is operational, tests will be made where data is transmitted between the MIDlet and the server to make sure everything functions as desired. The testing will go on including storing the sent data in the database and retrieving data for transmission from the database.

After that, the overall structure of the website will be set together with a proper database connection. A simple map will also be assembled which later will be extended to show coordinates which are fetched from the database.

Coming this far the base of the entire system is set: the MIDlet has been created, the communication with the server and database has been established and the website has been erected. What is next is to retrieve GPS data from the phone and send it on to the server. According to the wishes of Svep the work will mainly be focused on the GPS device developed by Svep in the beginning and later on it will be expanded to include the built in GPS of newer phone models as well. The retrieved GPS data needs to be processed so the relevant data is extracted from the data. So after the separation has been done the relevant GPS data will be sent to the server for database storage. When implementing this section a number of tests will run simultaneously to make sure that the MIDlet works as required.

Being able to retrieve and send coordinates through the MIDlet is one aspect of the system, the other is the presentation of it. Thus the website will be expanded to

contain necessary user functions for presentation of user routes. These include among other; login, a user page with overview of relevant user data and a route page where a selected route is presented on the map.

Now all the main parts of the system are functioning and little by little components, modules and functions can be added to the MIDlet and website. A small demo can also be released for chosen persons from Svep for some user testing.

Needed details and add-ons which are not thought of in the beginning will probably reveal themselves by this stage together with bugs from the user tests and time will be spent to implement all needed extra functionality and resolve all bugs.

Once the system is stable and works satisfyingly the support for using the built in GPS of newer phone models will be added to the phone. This will result in the MIDlet having a broad group of users: those with older phone models can buy the extra GPS device and those with newer phone models will have GPS built into their phones.

Having reached this stage the system is basically ready for being customized into a commercial product. Hence the final part will be to make a small study on how the system can be marketed commercially.

2.1 Project model

The project model followed during the development will be an agile method with development iterations throughout the life-cycle of the project. Since the project is an internal project at Svep and there are no real customers yet there will not be a proper documented requirements specification. Rather regular meetings will be held with supervisors at Svep where needs vs. difficulties on the way will be expressed and discussed. What is being developed next is thus decided together at the meetings. Once the decided functionality is implemented and tested new meetings will be held to decide the next step.

2.2 Research on the system and all relevant aspects

Due to the vastness of the project it is very important to have a clear picture of what is being developed, why it is being developed and to whom it is being developed for. The better understanding achieved when researching the background of the project the easier will the implementation be and the lesser is the chance of miscalculations when coding.

The main part of the research will be based on throughout discussions with involved parts at Svep.

2.2.1 The construction of the GPS device (HGE-100)

To be able to retrieve and use data from the HGE-100 GPS device smoothly it is necessary to understand how the device functions. Hence discussions with the developers of the HGE-100 GPS device will be taken as well as analyzing code snippets from demo MIDlets, to see how data is retrieved practically.

2.2.2 J2ME

The research will consist of reading up on J2ME and how MIDlets are structured, mainly from online material. Code snippets of smaller MIDlets previously developed at Svep will also be analyzed to understand how a MIDlet is built and works.

2.2.3 .NET framework

To gain an insight in Microsoft's .NET framework an introduction will be held by one of the Svep's developers. Following the introduction there will be some reading of .NET material. Finally a couple of tutorials will be used to implement some demo pages. The introduction along with the reading and practical examples through the tutorials will give a good overview of how the .NET framework and all its components work and interact.

2.3 Designing the database

In order to design the database correctly without having to alter the base of it when modifications are added in the future it is essential to have done a proper research on what the system is used for and at whom it is targeted.

This will be achieved through some brainstorming with involved parts at Svep as well as letting an experienced co-developer give feedback while designing the database.

2.4 Constructing the main structure of the MIDlet

Firstly a suitable development environment will be set where there is contact between the mobile phone and the computer for live debugging over Bluetooth.

Thereafter a main structure of the MIDlet will be built consisting of a simple menu system and a few screens which are shown when the user makes appropriate selections.

2.5 Establishing a network connection between the MIDlet and the server

Since a little application for transmitting data between the cell phone and a server already has been coded by Svep for testing purposes, the code will be analyzed to get a picture of how it works.

The code will be integrated into the MIDlet and then extended to include database interaction on the server side based on the transmitted data from the client side (the MIDlet in the cell phone).

2.6 Constructing the main structure of the website

The main structure of the website will be constructed in such a way that all files only consist of the body while the static sections of the website, such as the header, the sidebar and the footer only are included. In such a way if for example the header is to be altered, only one file needs editing which simplifies modifications of the site significantly.

A database connection will be established as well. Apart from this a simple map is also compiled. Features which are available from the programming interface of the map provider will also be implemented and tested to see what possibilities there are in the presentation of a route. The coordinates put on the test map will first be statically chosen and once everything works they will be fetched dynamically from the database.

2.7 Retrieving GPS data from the GPS device

The code available on Svep to retrieve GPS data from the HGE-100 GPS device will be merged into the MIDlet. Thereafter the MIDlet and the merged code will be modified to work together as a single code.

Once the GPS data is retrieved the data will be processed, formatted and put into local variables in the MIDlet for further use, such as displaying what the current location is on the screen or transmission to the server.

2.8 Transmitting the GPS data

Since a network connection has been established between the MIDlet and the server, the MIDlet only needs to be adjusted to send the local variables holding the relevant GPS data.

The server side will be coded in such a way to recognize when it is GPS data are received. The received GPS data will be processed and then stored in the database.

2.9 Presenting the GPS data on the website

The coordinates will be dynamically fetched from the database and presented on a map. A line should be drawn through the coordinates to better illustrate the route. Some relevant facts about the route, such as the covered length, the time and the like should also be presented.

2.10 Adding functionality to the MIDlet

When coming this far in the development it will be revealed what kind of features are lacking in the MIDlet. Hence at this stage needed extra functionality should be added to the MIDlet to increase the user satisfaction and experience.

2.11 Adding functionality to the website

As the website is the central point of the entire system from a user perspective it is important that there are enough features for the user not to feel disappointed. Thus it is of great matter to add functionality which is needed and appreciated by the users. Some time should also be spent on the design of the website to make it delightful for the eyes to rest on. However it is also important not to spend too much time at this stage neither since a lot will probably change when it is known how the system is going to be marketed and used as a final product.

2.12 Adding support for built in GPS

In order to broaden the group of users it is critical to add support for built in GPS in the MIDlet. Built in GPS is becoming common in newer cell phones and it is only a matter of time before it is available in the almost any new cell phone. Consequently support for built in GPS devices will eventually make it possible for practically every user who owns a cell phone to use the system.

2.13 Marketing the system

As a final part a study will be presented with areas of use and marketing of the system. This study will mostly be based upon brainstorming and discussions with involved parts.

3 Implementation

The implementation of the system pretty much followed the methodology.

3.1 Research on the system and all relevant aspects

The research on the system began with a couple of throughout discussions with involved parts at Svep. The discussions helped to get a clearer picture of what is being developed and why it is being developed (in other words what it is meant to be used for).

The initial idea was to create a system where on one hand a user through his cell phone using a GPS device could send his coordinates as he was moving along his route and on the other hand the user's family and friends could keep a track on him on a map. Hence the social aspect of the system played an important role which gives the system a community-like feeling.

How this was to be realized practically was not known but after discussions it was decided that there should be some kind of access system where only people with authority are allowed to watch a route. It is the owner of the route who grants access to his routes. Hence some sort of "Friends" list is needed from which the user chooses who is authorized to view his route.

3.1.1 The construction of the GPS device (HGE-100)

Speaking with the developers at Svep revealed that their developed HGE-100 GPS device provides NMEA data every second once initiated. NMEA is short for "National Marine Electronics Association" and is a combined electrical and data specification for communication between many marine electronic devices and instruments, among those GPS receivers. The NMEA standard uses an ASCII serial communications protocol where all necessary location data can be found.

Svep provided with a small code sample of how NMEA data can be retrieved from the HGE-100 GPS device. The code sample was analyzed to have a greater insight in how the HGE-100 GPS device works.

3.1.2 J2ME

For software development on smaller devices such as cell phones and PDAs Java has provided with a collection of application programming interfaces (API) commonly known as J2ME. Even though the construction of MIDlets in J2ME differs a bit from computer applications still developing in J2ME is very similar to coding Java for computer applications.

The construction and structure was studied to understand how different components in a MIDlet interact and when different methods are invoked. A few samples and tutorials were studied and implemented as well to get a taste of how J2ME practically works.

3.1.3 .NET framework

Since Svej uses the Microsoft server environment it was desired that the website be developed in the .NET framework. With its framework Microsoft has succeeded in combining a very comparative development kit where database design, graphical user interface (GUI) design and a programming interface interact in an exceptionally smooth manner.

The first step in getting a grip of the whole framework was to obtain information about the different mechanisms behind. Thereafter a few test pages were created to experiment with some of the manifold components available. When it comes to developing websites (where ASP.NET is used) the .NET framework provides a neat feature. Each ASP file (*.aspx) has a “code-behind” class file (*.cs) for full scale programming, usually in the VisualBasic (VB) or C-Sharp (C#) language. Hence in the class file advanced methods and database inquiries can be performed and then presented in the ASP file which mainly handles the visible part of a webpage.

Consequently as a final step in the research on the .NET framework the created test pages were coded dynamically through the class file. This gave an insight in the relation between the class file and the ASP file and how powerful website can be created.

3.2 Solution overview

Based upon the research an overview was drawn to have a better comprehension of how the system would be realized:

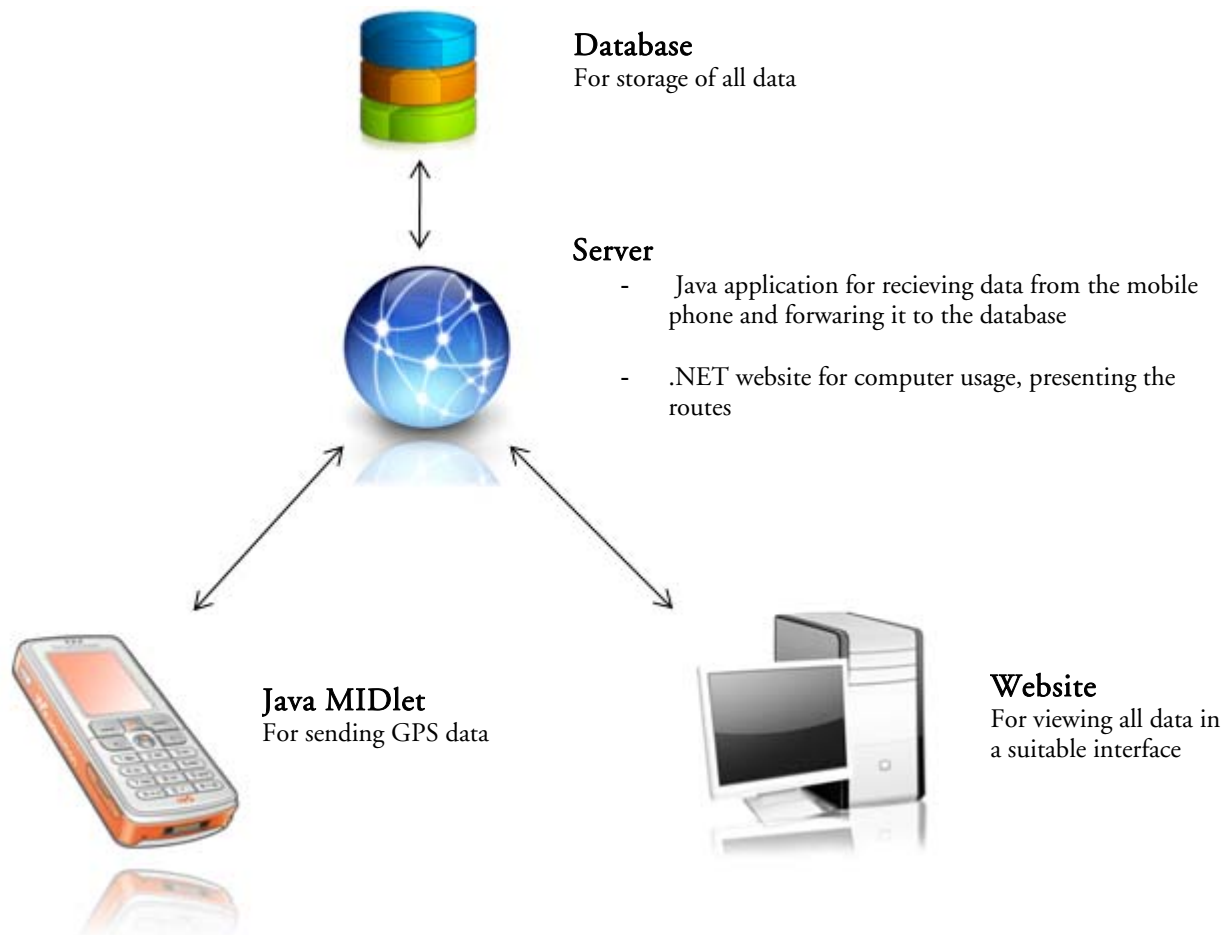


Figure 1: An overview of the entire solution

3.3 Designing the database

Since the database is the central connecting point of the entire system it is crucial that it is designed in such a way that no alterations are made to its core later on. Hence the tables and their relationships should be outlined cautiously while having in mind that further tables might be added in the future, but these added tables should not interfere in the prevailing structure.

Thus, after some careful consideration, the core of the database was summed into four tables: Users, Friends, Routes and Coordinates.

As the names suggest the “Users” table is dedicated for all user data, such as username, password, email and the like.

The “Friends” table keeps hold of which users have a friendship with each other.

The “Routes” table contains information on all user routes. This table contains a unique route id for each route, which user is the owner of the route along with route name and description.

Finally the “Coordinates” table holds all relevant location data, such as longitude, latitude, altitude, timestamp and the like. There is also a column dedicated to messages. The messages column is set to null as default but if a user sends a message then the message is stored in this column on the specific coordinate.

Each table has its unique id as a primary key which is used when creating relations between the different tables.

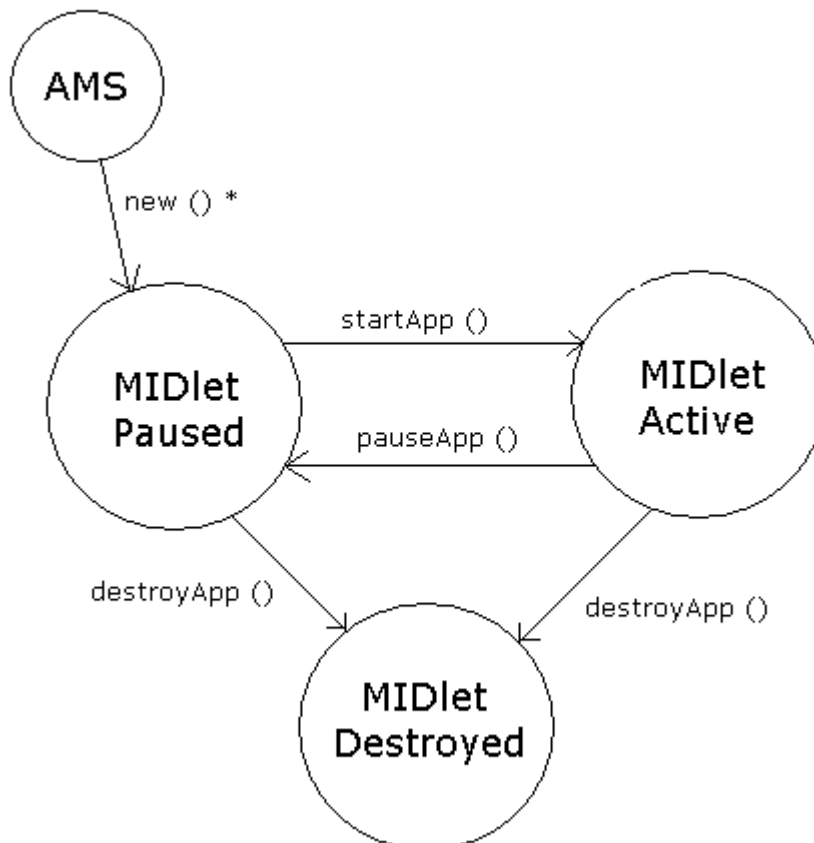
3.4 Constructing the main structure of the MIDlet

This section explains how a MIDlet generally works as well as explaining the first developed screens for the GPS Tracking System MIDlet.

3.4.1 How a MIDlet works

To develop a properly functioning MIDlet it is important to understand how a MIDlet works.

All J2ME MIDlets contain a class which extends the `javax.microedition.midlet.MIDlet` class. Unlike ordinary Java applications which have a “main” method a MIDlet runs in a particular environment where its functioning is controlled by the platform’s Application Management Software (AMS). A MIDlet can be in three states: an active state, a paused state and a destroyed state.



* - creates new MIDlet instance using MIDlet's no argument constructor

Figure 2: Illustrating the states of a MIDlet

The “startApp()” method is instead similar to the ordinary Java “main” method which is the first method to start executing when the MIDlet is started.

The “pauseApp()” method is a invoked when the MIDlet goes into a paused state in order to release resources.

And finally the “destroyApp()” method is called when the MIDlet is about to be shut down. In this method all final needed code can be executed for a clean shut down, such as closing open streams, saving data and so on.

Thus the main structure of a functioning J2ME MIDlet looks like this:

```
public class Demo extends MIDlet {  
    public Demo() {  
    }  
    protected void destroyApp(boolean unconditional) {  
    }  
    protected void pauseApp() {  
    }  
    protected void startApp() {  
    }  
}
```

3.4.2 J2ME User Interface Architecture

J2ME use the javax.microedition.lcdiui package to create user interfaces for MIDlets. There are a manifold of elements which are displayable on the display. These are of two kinds, high level and low level. High level elements have already been structured and only need to be customized as desired. Low level elements allow manipulation and drawing of low level graphics such as drawing of shapes, forms, images and the like.

The display in the lcdiui package can only display one displayable object at the time. If a new object is displayed the old one isn't destroyed and can be displayed again. To display an element the “setCurrent()” method is invoked.

The tree below gives a good overview of the “Displayable” package:

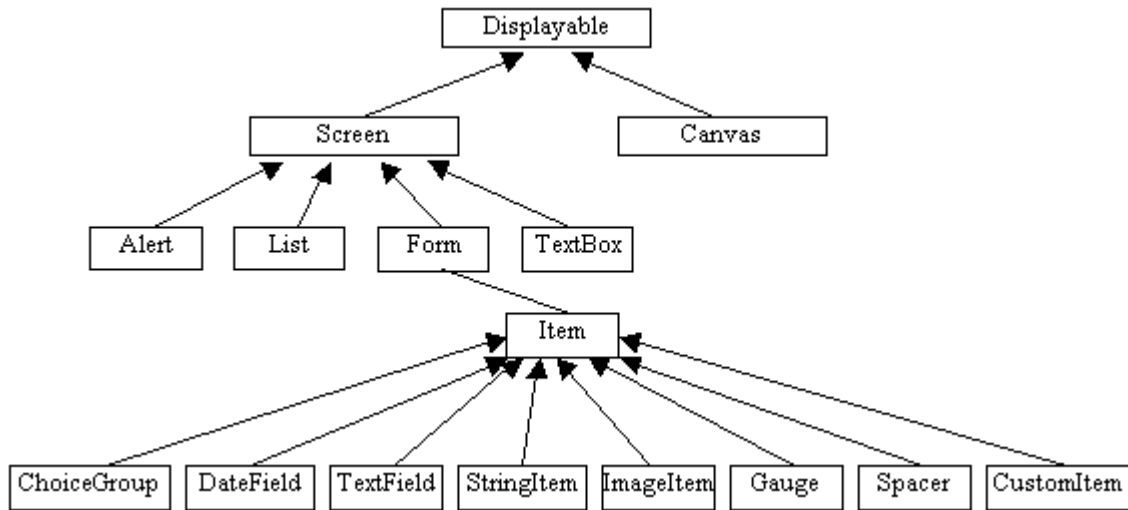


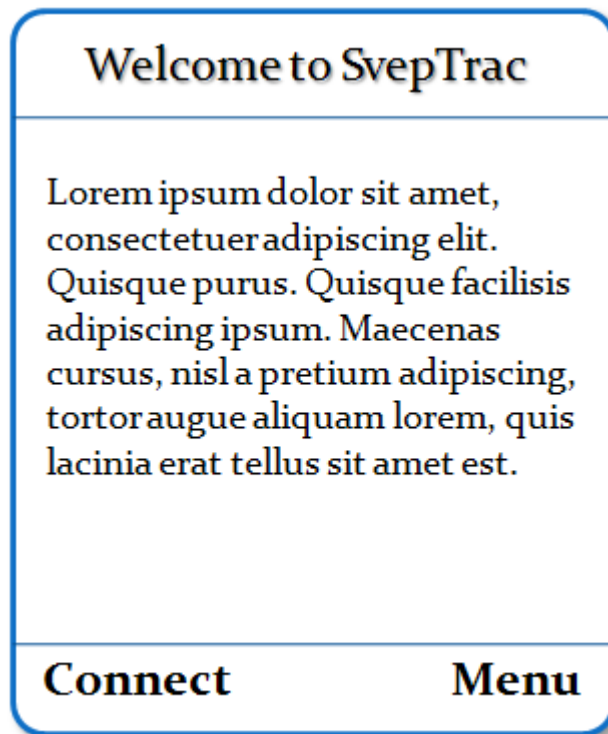
Figure 3: Illustrates an overview of the “Displayable” class

Thus a display can show either a graphics object called “Canvas”, where all the user interface is drawn, or a screen object consisting of either an alert, a list, a form or a textbox. A form in itself is empty and can consequently be filled with “Item” elements.

3.4.3 Paper prototypes of the different MIDlet screens

Before creating the first screens of the MIDlet paper prototypes of all the main features of the MIDlet were sketched to get a clearer picture of how the MIDlet should look and function. The paper prototypes were also shown to involved parts at Svep for feedback before starting the actual implementation.

The figures below illustrate the first two screens which the user at met with when launching the MIDlet:

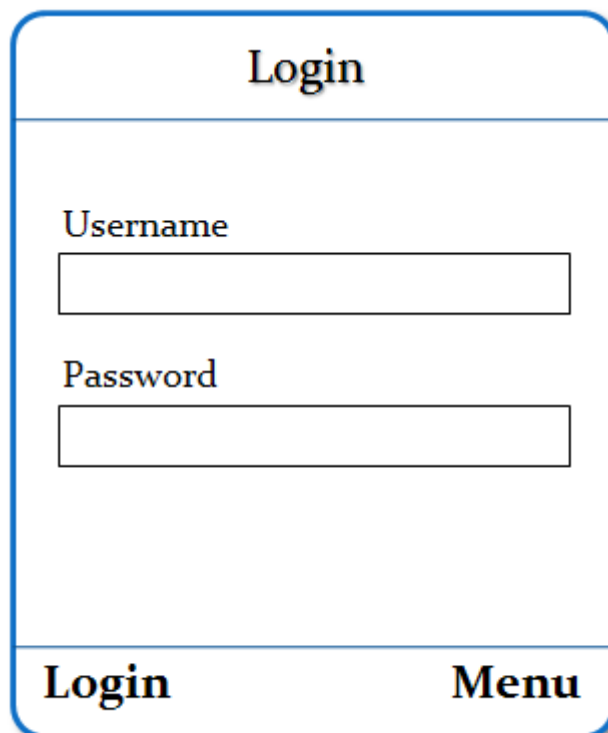


Welcome to SvepTrac

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque purus. Quisque facilisis adipiscing ipsum. Maecenas cursus, nisl a pretium adipiscing, tortor augue aliquam lorem, quis lacinia erat tellus sit amet est.

Connect **Menu**

Figure 4: Illustrating a welcome screen with the option of connecting to the server



Login

Username

Password

Login **Menu**

Figure 5: Illustrating the login screen which the user is met with after having successfully connected to the server

3.4.4 Implementing the first screens

As a main structure for the MIDlet a couple of forms were created, based upon the paper prototypes, with simple commands appended to them. The high level design was used since it is easily customizable and works on all platforms, even though it is very limited in terms of appearance. The design and appearance of the MIDlet is instead saved for the future when it is decided how the system is going to be marketed. To put time and energy on the design at this stage would only be a waste of time since the design will probably change later on and also a lot of time must be spent on smaller adjustments and details in order for the design to work properly.

The first form was a presentation screen with a welcome message and the instruction to press “Connect” in order to establish a connection with the server and start using the MIDlet. The actual connection was to be implemented when a network connection had been established between the MIDlet and the server. The next screen was a login form with two text fields to enter username and password along with a “Login” command. When the login command was pressed and text had been entered into the text fields the idea was that the entered strings would be fetched from the text fields and sent to the server for validation. Again the implementation of the actual server connection was saved for later when the network connection had been established.

3.5 Establishing a network connection between the MIDlet and the server

To establish a mutual network connecting, a java application was developed for the server side, referred to as server handler. The server handler is basically a thread listening to incoming connection requests on a specified port using Java’s “ServerSocket” class. Each time a MIDlet tries to connect to the server using the IP number and socket port specific for the server the server handler detects the incoming request and starts up a new thread handling the connection and data transmissions. When the MIDlet is disconnected the thread is also closed. All attempts and errors are also logged in a file as a point of reference.

The server also needed to be somewhat configured and necessarily had to be opened in order for the connection to work. This was a bit of hassle though since when the ports were opened by the network administrator at Svep they were only opened for internal traffic. Hence when the MIDlet, coming from the cell phone’s external internet connection, tried to connect to the server it could not find any open port on the specified IP number. When the port was pinged using the command prompt it replied with an affirmative though, since the ping was done through the computer which accessed the internet through Svep’s intranet. When pinging from the computer the traffic was categorized as internal and hence the server responded, but when the cell phone tried to connect it was classified as external traffic and consequently was rejected.

It took a while to discover the mistake but once it was spotted it only needed some minor adjustments to work. The result was that the server handler now detected when cell phones tried to connect to the server and accordingly established a mutual connection.

Once the connection was established the next step was to transmit data. For the MIDlet's connection to the server the common "StreamConnection" class provided by Java was used. The "StreamConnection" class has output streams and input streams to handle outgoing and incoming data. So the MIDlet's "OutputStream" was directed at the server application's "InputStream" and the server's "OutputStream" was directed at the MIDlet's "InputStream". Since the connection was established the streams were also ready for traffic.

As a first test the square symbol (#) was sent from the MIDlet using the output stream. The server handler, which is a running thread blocked from executing until data is available in the input stream, was coded to receive all data and print it to the command line, which it successfully did.

Now it was time to send back data to the cell phone. Hence a feature was added to the server handler to compare the incoming data to a specific string (in this case the specified string was "STX") and if there was a match then it would reply with a square as well. The MIDlet was configured to print out all incoming data on Eclipse's console line using the "System.out.println" method. Instead of the MIDlet sending a square it instead sent the specific command ("STX"). The server handler successfully received and detected the match of the string but the MIDlet never received the reply which the server handler tried to send.

After some investigations it was found out that when reading the input stream a time out of 300 seconds had been set, however the logical statement to verify if the waiting time had reached time out was not correct and hence the MIDlet was looping in wait mode for 300 seconds! This was fixed and the MIDlet managed to receive and print out the square.

Having reached this far the next step was to establish a connection between the server handler and the database. Therefore another class on the server side was added to the server handler to manage database inquiries, referred to as database handler.

Instead of sending a random symbol the MIDlet was coded to send the username and password entered in the login form on the MIDlet. The username and password was set together in a string with a specific command at the beginning so the server can detect that the incoming data is login details. Therefore the server handler was also set to compare the beginning of incoming data with the chosen login command and if

there was a match then it would parse the username and password and forward it to the database handler.

The database handler then would run an SQL inquire in the database. If a user with the specified username and password existed it would return the user id of the user to the server handler, otherwise it would return 0. The server handler would return the user id to the MIDlet. If a 0 was received the user would be prompted with an error alert saying that wrong username or password had been specified and that he should try and login, otherwise the user would be taken to the next screen.

This step was fairly trouble-free and a proper database connection was now established.

Other developers at Sveg who used the same server noticed that there was a processes running in my name which used maximum CPU usage. For some reason the server handler would use all computer resources each time a cell phone was connected to the server. After some troubleshooting it was concluded that the problem was caused by the server handler's infinite while loop where it was waiting for data from the input stream. Instead of using the input stream's read() method, which blocks the thread from executing until incoming data is available, the available() method was instead used which either returns the number of available bytes in the input stream or -1, but it does not block the thread from running. In other words the 'While-loop' kept executing the available() line over and over again, consuming all CPU resources. This was solved through adding a sleep for a couple of hundred milliseconds which provides plenty of time for the CPU to execute other processes in the mean while.

3.6 Constructing the main structure of the website

On a website there are a couple of elements which usually do not change content or changes very little, yet they are included in all pages on the website. These elements are the header, the sidebar and the footer. If these three elements would be coded from new in each page on the website, a slight modification in for example the footer would result in changes in all pages. Instead the main structure of a website should be dynamic. This can be achieved through mainly two methods. One method is to split up the page in divisions where each division has a file included:

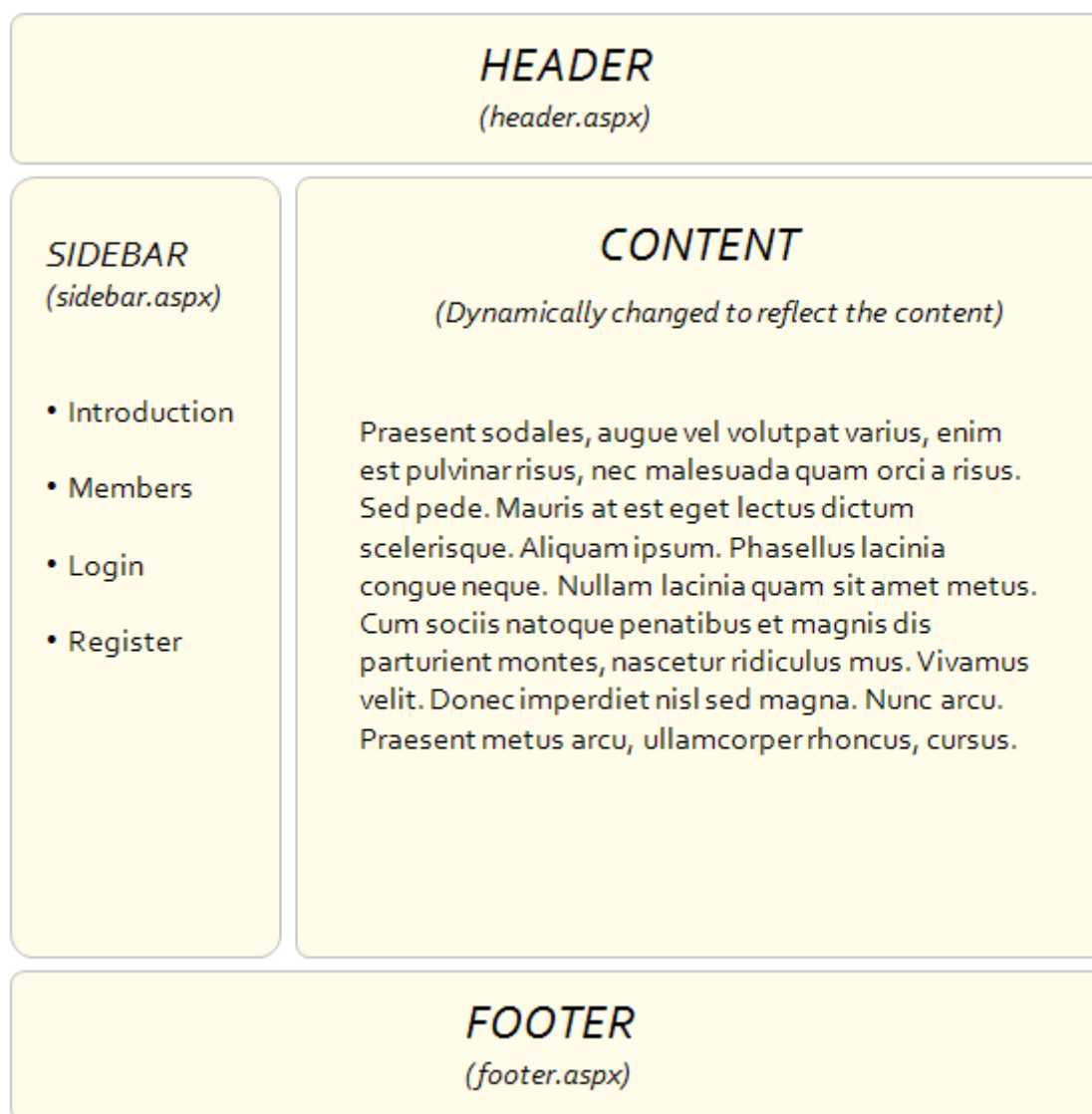


Figure 6: The structure of a master page in a website

As picture 2 suggest three files, header.aspx, sidebar.aspx and footer.aspx, are created and filled with relevant content. These three files are then included in the every content page. Hence if modifications are needed in for example footer then only one file (footer.aspx) is altered and the changes will be reflected on all pages.

When it comes to the second method Microsoft’s Visual Studio, which is the primary software used to code in the .NET framework, has a preinstalled template called “Master Page” to accomplish a dynamic website structure in a slightly different way. The “Master Page” contains all necessary parts of a page except the actual content. Where the content is to be displayed instead an ASP.NET component called “ContentPlaceHolder” is used in the following manner:

```
<asp:ContentPlaceHolder ID="MainContent" runat="server">
</asp:ContentPlaceHolder>
```

Now all other pages, instead of being independent, have their content placed in the following tag:

```
<asp:Content ID="mainContent" ContentPlaceHolderID="MainContent"
runat="server">

    <!-- All content which comes here is later dynamically
shown in the "ContentPlaceHolder" described above. -->

</asp:Content>
```

When a visitor visits a page, for example “MemberList.aspx”, which is structured in the above mentioned way the server will compile “MemberList.aspx” in such a way that the “Master Page” (containing the header, sidebar and footer) is merged with “MemberList.aspx” and they appear as one single page to the visitor.

In the beginning the first method, which is also very common in other web application platforms, such as for example PHP, was used. There was one negative aspect with this method though, which was realized later on. Each page must follow a common structure when splitting up the areas. So if each page for example reserves 200 pixels in width for the sidebar and later on it is decided that the sidebar width must be decreased to 150 pixels, it means that all content pages must be changed to reflect the new width.

Hence after some consideration the main structure of the website was swapped to use the second method. Everything concerning main structure and appearance is then restricted to the “Master page” and changes only need to be in one file.

A CSS file was linked to the website where all formatting and design (such as size and colours of different divisions, text appearance and the like) could be set. A simple menu with the most necessary links, like a profile, a members list and a routes page, was created (the actual address for the links was to be added later once the pages were created). Also a simple header and footer were produced to fill the areas.

3.6.1 The login module

As a part of the main structure a login module was created. The difficulty behind the login module was to establish a proper and secure authenticated session which enabled the user to access restricted sections of the website.

There were a few approaches to solve the authentication problem. One was to store the user’s password in an encrypted form in a cookie (a text file on the user’s local computer. Each time he accesses a restricted page the stored password will be compared to the password in the database and if they match he is given authorization to view the page. If the user logged out then the password variable would simply be cleared. This method was not appreciated since storing data on the local computer might be fine as long as the user is using his own computer, but once he is elsewhere the risks of privacy violations increase.

Instead something called session variables were used. Session variables are variables which are stored at the server and are valid for the current session. How long a session lasts can be set in the configuration file of the website in Visual Studio. As a start the session was set to last for 60 idling minutes, meaning if the user is inactive for more than 60 minutes his session is terminated and consequently the variables are cleared and he is logged out.

3.7 Retrieving GPS data from the GPS device

To retrieve GPS data from the SonyEricsson HGE-100 GPS device is fairly simple. The data is encoded in NMEA messages and hence needs to be parsed in order to get the relevant data.



Figure 7: Shows the SonyEricsson HGE-100 GPS accessory

3.7.1 Establishing a connection

When the GPS device is connected to the cell phone a virtual serial channel is created. The channel is visible as a com port with the name "AT5" and the baud rate 9600. Thus to establish a connection with the com port the following line of code is used:

```
CommConnection commConn =  
(CommConnection)Connector.open("comm:AT5;baudrate=9600");
```

The following lines open the input and output streams to and from the HGE-100 GPS device:

```
InputStream iStream = commConn.openInputStream();  
OutputStream oStream = commConn.openOutputStream();
```

3.7.2 Controlling the data flow

Once the connection is established and the streams are opened it is possible to start or stop retrieving NMEA data using two special commands:

```
/*
```

```

* Send the commands to HGE-100.
* - $STA will result in a stream of NMEA data,
*   4 messages each second.
* - $STO will halt the stream of data
*/
String cmd = "$STA\r\n";
oStream.write(cmd.getBytes)
cmd = "$STO\r\n";
oStream.write(cmd.getBytes)

```

When the start command is sent to the HGE-100 GPS device it is possible to retrieve the NMEA data by reading from the input stream. The device will then return new NMEA data every second.

3.7.3 Parsing the NMEA data

The following section will explain the content of the NMEA data and how it was treated in the system.

3.7.3.1 Analyzing the NMEA data

The NMEA data is encoded as an ASCII string with commas to separate each word. The string starts with a dollar sign (\$) and ends with an asterisk (*). A string of data looks like this:

```

$GPRMC,214434,A,3753.666,N,12203.162,W,0.0,0.0,270901,15.4,E,A*33
$GPGGA,214616,3753.667,N,12203.167,W,1,04,5.6,121.1,M,-27.4,M,,*7
$GPGSA,A,3,01,03,20,22,,,,,,,,,7.4,5.6,1.5*36
$GPGSV,3,1,10,01,69,062,47,03,12,106,37,04,12,279,00,08,12,250,0*77
$GPGLL,3753.667,N,12203.167,W,214616,A,A*54
$GPBOD,,T,,M,,*47

```

The abbreviations in the string imply the following:

\$GPRMC: Recommended Minimum Specific GPS/TRANSIT Data

\$GPGGA: Global Positioning System Fix Data

\$GPGSA: GPS DOP and Active Satellites

\$GPGSV: GPS Satellites in View

\$GPGLL: Geographic Position, Latitude/Longitude

\$GPBOD: Bearing, Origin to Destination

For the GPS tracking system the \$GPGGA sentence is needed since it contains the relevant GPS data. The \$GPGGA sentence contains the target's topological location information. The relevant fields are as follows:

Field 1: Message Header (\$GPGGA)

Field 3 – 4: Latitude, North/South

Field 5 – 6: Longitude, East/West

Field 10: Altitude in meters

A \$GPGGA sentence looking like this:

```
$GPGGA,214616,3753.667,N,12203.167,W,1,04,5.6,121.1,M,-27.4,M,,*7
```

Means that the latitude (given as 3753.667) is 37 degrees, 53 minutes, and 66.7 seconds.

3.7.3.2 Treating the NMEA data

Once the \$GPGGA sentence is retrieved from the ASCII string it is passed on to a method where each field in the sentence is extracted, using Java's string specific methods, and then stored in its corresponding variable. In other words the latitude field is extracted and stored in a variable called "latitude", the longitude field is extracted and stored in a variable called "longitude" and similar. The fields which are extracted are latitude, longitude, altitude and the timestamp.

These variables are then used when sending the GPS data to the server for database storage.

3.7.4 Integration into the MIDlet

Support for the HGE-100 GPS device was integrated into the MIDlet by creating a thread where NMEA data was read from the GPS device. After a connection had been established to the GPS device the thread was started where the NMEA data was read from the GPS device's serial port. Once the NMEA data in the form of an ASCII string was read the \$GPGGA sentence was extracted and sent on to a method where the fields containing relevant GPS data were put in their own variables (i.e. the latitude was put in a latitude variable, the longitude was put in a longitude variable and so on).

This thread runs in a one second interval where fresh GPS data is provided by the HGE-100 GPS device every second if satellites are available.

3.8 Transmitting the GPS data

In order to transmit the GPS data from the MIDlet to the server the local variables where the GPS extracted GPS data was stored are combined into a single string. This string has a unique start command and separating character between each variable. The string is transmitted to the server using an output stream in the following manner:

```
// Send a message to the server
// "request" is the combined string containing the GPS data
os.write(request.getBytes());
os.flush();
```

On the server side the string is received through the input stream's read() method:

```

byte[] data = new byte[1000];
int length = is.read(data);
byte[] truncData = new byte[length];
System.arraycopy(data, 0, truncData, 0, length);
String message = new String(truncData);

```

The “message” string which contains the transmitted GPS data string is then checked and if the initiated command is equal to the command set for transmission of GPS data the “message” string is forwarded to a local method where the embedded GPS data is extracted from the string and stored in their own variables. The variables are then sent on to the database handler which stores them in the database through an insert SQL query.

3.9 Presenting the GPS data on the website

To present the GPS data on the website the Google map service was used. The reason for choosing Google Maps was partly due to it being free, fast and reliable and partly due to it having an extensive API where all sorts of features and functions can easily be implemented by the developer. Thus, instead of only showing the route on a map, there is the possibility to enhance the presentation by adding extra functionality as well as being able to meet future demands and wishes.

To implement a Google Map on a personal website is rather simple and there is plenty of documentation and tutorials as a help on the way. The API is in JavaScript and hence all coding is done in JavaScript as well. Since the GPS data which is going to be shown on the maps are fetched dynamically from the database it is also important that the maps are coded dynamically. Hence all map coding is done in the “code-behind” class file.

The first thing to do is to create the actual map along with desired map controls:

```

//Creating map with toolbars
var map = new GMap2(document.getElementById('map'));

//Scaling and repositioning toolbar
map.addControl(new GLargeMapControl());

//Toggler for switching between the Google map and satellite views.
map.addControl(new GMapTypeControl());

//Map overview
map.addControl(new GOverviewMapControl());
//It is necessary to make a setCenter call
//of some description before adding markers
map.setCenter(new GLatLng(0,0),0);

```

The lines of code above will create and show the map in a div called “map” which can be placed anywhere on the website simply by writing:

```
<div id="map"></div>
```


The object containing coordinates which is used in the API is called “GLatLng” and takes two input parameters; latitude and longitude. Since a map usually contains several coordinates the Google Maps API also has a sort of vector where all points are stored called “GLatLngBounds”.

Thus the map was implemented in such a way that, after having created the map, an SQL inquiry is done where the coordinates of a specific route are fetched. The fetching is done in a loop where the coordinates are stored in C# variables. The C# variables are then used to create a new “GLatLng” object which is appended to the “GLatLngBounds” coordinates vector. When there are no more coordinates left the loop will end and all coordinates have been stored in the “GLatLngBounds” coordinates vector.

The hard part of retrieving the GPS data from the database and present it on the map is to successfully combine the two programming languages C# and JavaScript. Syntax errors are easily generated but not so trivial to discover and mend. It does take a while to master coding in both languages simultaneously and make them interact but once it has been mastered it is a quite powerful developing technique.

The next step was to visualize the route by drawing a line through all coordinates. Google Maps provide a function called “GPolyline” for this as well. There are two ways to draw a line through all points; the easy way and the almost as easy way. The easy way is to simply write:

```
//The variable "points" is a vector containing
//all points which a lines is to be drawn through
var polyline = new GPolyline(points);
map.addOverlay(polyline);
```

The almost as easy way is pretty similar with the difference that instead of drawing a line through all points, a line is drawn between two neighbour points while looping through all points. So instead of having one long line several point-to-point lines are used to create a chain like line through the entire route:

```
for (var i=1; i<points.length; i++){
    var polyline = new GPolyline([ points[i-1],(points[i]) ]);
    map.addOverlay(polyline);
}
```

At first the easy method was used, however it was discovered at larger routes (containing more coordinates) that the line disappeared after zooming in on the map. Hence the second method was adopted where it was possible to zoom in to a greater extent.

3.10 Adding functionality to the MIDlet

Coming this far the skeleton of the system had been erected and the time had come to start adding some useful functionality. Since the system is a prototype only the core features needed to create a handy MIDlet was implemented.

3.10.1 List and create routes

Since coordinates are categorized by routes it is essential that the user has the ability to choose which route to send his coordinates to. Hence a function was implemented where route data containing route name and route id was retrieved from the database, sent from the server, received by the MIDlet and presented in a list for the user.

The steps to accomplish this task were fairly simple. First a special command was sent from the MIDlet to the server. The server was set to execute a select SQL query when the special command was received. The returned route data from the database was then merged into a single string object with a particular symbol separating each route. Afterwards the string was sent back to the MIDlet which parsed the string and added each route to a list that finally was presented for the user. Behind the visible list each list element had the route's unique id connected to it and when the user chose a route its' id was set as the current active route id. This id was then used to store coordinates with the right route in the database.

It did not take long to realize that only listing prevailing routes was not sufficient as there might arise a situation where the user needs to create a route through the cell phone. If for example a person is outside in the woods wishing to map his walk it would be great if he could create a new route on the spot and then start storing coordinates under the new route.

The need was there and the realization did not take more than two text boxes and a couple of validity checks. One text box was mandatory and contained the route name and the other one was the optional description which could be added by the user. The validity checks just made sure there was some input entered for the route name. Once the necessary information was entered, validated and the user had pressed "Create" all data was merged into a string and sent to the server. The server forwarded the string to the database which inserted the route and returned the newly created route id which was sent back to the MIDlet. The received route id was then used as the id for the current active route so all coordinates which were sent were stored under the new route (until the user changes to another route).

3.10.2 Send messages

The ability to send a message was suggested already in the initiating stage of the project. Messages would work as a point of reference bookmarking or tagging certain areas with a descriptive text by the user. They could also be used by the user to notify his viewers who follow his route in real time what he currently is doing.

To implement the function a menu was added to the screen where GPS coordinates are shown in an ongoing route. To the menu a “Send message” element was added. Once chosen a new screen would pop up containing two text boxes: the message title and the actual message. The user could then enter a title for his message and a descriptive text. When “Send” is pressed the message is attached and sent along with the next set of coordinates. The server receives the coordinates, parses the data and inserts it into the database.

The next step was to visualize a message on the map. Google Maps has an object called “GMarker” which basically adds a marker to certain coordinate on the map. A “GEvent” listener can be added to the “GMarker” which creates a popup info window when the “GMarker” is clicked. Furthermore the standard icon for the marker can be replaced with another.

The approach for realization was to add a check when looping through all coordinates that if a certain coordinate has a message attached to it then a “GMarker” object was created with a “GEvent” listener containing the message text. The “GMarker” itself was named after the message title.

```
// === Create an associative array of GIcons() ===
var gicons = [];
gicons['message'] = new GIcon(G_DEFAULT_ICON,
    './images/map/message.png');
gicons['message'].shadow = './images/message.shadow.png';
gicons['message'].iconSize=new GSize(32,32);
gicons['message'].shadowSize=new GSize(59,32);
gicons['message'].iconAnchor=new GPoint(13,25);
gicons['message'].infoWindowAnchor=new GPoint(16,0);

//Function to create an info text box on markers
function createInfoMarker(point, label, msg) {

    var marker = new GMarker(point, gicons['message']);
    if (msg!=null)
        GEvent.addListener(marker, 'click', function() {
            marker.openInfoWindowHtml(msg);
        });

    return marker;
}

//Putting the marker on the map
map.addOverlay(marker);
```

3.10.3 Take photos

When the feature to send message was implemented it there were a series of discussions involving brainstorming and marketing ideas with all concerning parts at Svep. It was decided that mere mapping coordinates and sending messages wasn't sufficient for the system to stand out and be competitive to its rival counterparts. There needed to be something else which would really put the system ahead of others, and that was the ability to take photos. It was argued that if we manage to take and send photos with

the MIDlet and do it good then we would have a good head start in front of other concurrent.

To take photos at first was deemed to be rather complex. Thus it was determined that a study first should be made with an approximation of how long it would take to implement it and if it was a reasonable amount of time then the actual coding would start.

After a day or two of study, research and analyzing code examples I concluded that the support for photos was indeed achievable within an appropriate time span. After having received green light from the project manages the coding was initiated. The start and the end result covered two radically different approaches.

3.10.3.1 File attachment

At first a file manager was created. The idea behind the file manager was to let the user browse the photos he wanted to send in a message. Practically it meant that the user would minimize the MIDlet, take a photo and save it to his memory disk, resume the MIDlet and then attach it through the file manager.

The file manager was worked in such a way that it opened a file connection to the cell phones local drive using the following line of code:

```
/*For camera directory (this can be changed later to be dynamical
where the path = which directory the user chooses)*/
private String initPath =
    System.getProperty("fileconn.dir.photos");
//private String initPath = "file:///c:/";

FileConnection fc = (FileConnection) Connector.open(path);
```

In order to easily list all folders and files the file manager class was created as an extension to a list. In order to add an item to a list calling “append()” would do the job. To differ between a folder and a file an icon was also appended before the folder or file name. Once the stream was opened the directories and files of the specific path could be listed by writing:

```
// Get a filtered list of all files and directories.
// True means: include hidden files.
// To list just visible files and directories, use
// list() with no arguments.
Enumeration filelist = fc.list("*", true);

while(filelist.hasMoreElements()) {

    fileName = (String) filelist.nextElement();
    //fc = (FileConnection) Connector.open("file:///c:/" +
    fileName);
    fc = (FileConnection) Connector.open(path + fileName);
```

```

if(fc.isDirectory()) {

    setSelectCommand(cmdOpen);

    try{
        append(fileName, Image.createImage("/folder.png"));
    }
    catch (Exception ex){
        append(fileName, null);
    }
}
else {

    setSelectCommand(cmdAttach);

    try{
        append(fileName, Image.createImage("/image.png"));
    }
    catch (Exception ex){
        append(fileName, null);
    }
}
}
fc.close();

```

When a user selects a file the content of the file is placed in a byte array and then sent along with the message to the server. The server receives the data, creates a new file with the given file name and opens a “FileOutputStream” to the created file into which it directly streams all the file’s byte array data read from the input stream.

A few difficulties were met during the process. It took some time and testing before the correct path could be found to open a file connection to. Another problem was how to split up the data into smaller packages and send them without breaking the file structure. After some testing and careful file handling the following setup, sending the file in packages of 1000 bytes each, was proved to work best:

```

if(file!=null){

    try{
        int sent = 0;
        while (sent<file.length){

            if( (sent+1000)<file.length ){
                os.write(file, sent, 1000);
                os.flush();
                sent += 1000;
            }

            else{
                os.write(file, sent, file.length-sent);
                os.flush();
                sent = file.length;
            }
        }
    }
    catch (IOException ioex) {

```

```

        System.out.println("Send File I/O Error: " +
                           ioex.getMessage());
    }
    file = null;
}

```

The greatest obstacle however was to make the server understand when to finalize the created file and stop putting in data. At first the server didn't know when to close the file and hence all GPS coordinates which were sent after sending a file were stored inside the file resulting in a broken image file. When the issue on how and in what packages to send files were solved, it was also possible to cleverly code a properly functional receive method:

```

File file = new File(album + "\\\" + fileName);
FileOutputStream fos = new FileOutputStream(file);

for(int i=0; i<size; i++){
    try{
        fos.write(is.readUnsignedByte());
    }
    catch (EOFException eofex){ }
}
fos.close();

```

Principally what is done is before transmitting the file the MIDlet sends the file size which is received by the server. When the file transmission then is initiated it will only be allowed to go on as long as the number of received bytes doesn't exceed the file size! In this way the server knows when the file transmission is done and it can go on receiving coordinates as usual.

3.10.3.2 Photo capture

Once the file manager was done, fully integrated into the system and had been wholly tested new problematic issues arose. Firstly since the MIDlet isn't signed the user was forced to grant permission to each file which is listed. Having many files or folders would in other words result in a lot of button presses! Secondly since this approach doesn't let the user take photos directly from the MIDlet the user friendliness of the MIDlet is put to question. How to minimize and resume a MIDlet is not obvious and not everyone knows how to do it. To first minimize the MIDlet, take a photo, resume the MIDlet, browse for the newly taken photo and finally attach it is, besides being rather complicated, a major detour.

So after some careful considerations it was decided to integrate a photo control to take photos directly in the MIDlet. This at first seems a lot harder than it was. But at the end it all came down to a few lines of code where you first create the "VideoControl" object and then create a separate screen for it:

```

try {
    player = Manager.createPlayer("capture://video");
}

```

```

        player.realize();
        VideoControl videoControl = (VideoControl)
player.getControl("VideoControl");
        Item item = (Item)
videoControl.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
        formCamera.append(item);
        display.setCurrent(formCamera);
        player.start();
    }
    catch (Exception e) {
        System.out.println(e);
    }
}

```

The taken photos were then placed in a byte array, similar to sending a file, and then sent on to the server:

```

byte[] file = videoControl.getSnapshot(null);
        //videoControl.getSnapshot("encoding=jpeg&width
        =160&height=120");
        //videoControl.getSnapshot("encoding=png&width=
        80&height=60");
        //videoControl.getSnapshot("encoding=bmp&width=
        160&height=120");
player.close();

```

This technique works much better. A menu item was added to the screen where GPS coordinates are sent with the option to take a photo and once it was selected the video control screen would be set as the current screen. When the user takes a photo or aborts the operation the MIDlet will return to the previous screen (sending a message). This method made it possible to take a photo with two keypad presses and was thus greatly improved the user friendliness.

3.10.4 Offline mode

At one stage during the testing process one of the co workers at Svep was travelling to Tunisia for his vacation. It was thought that a couple of real scenario routes would be very handy when it came to marketing the system later on, and the trip to Tunisia was a perfect opportunity. There was one problem though: the high expenses to cover the international data traffic. With less than a week left to the journey it was decided to implement an offline mode for the MIDlet where the user could save his route locally on the cell phone without connecting to internet and later on upload it to his account through the website.

Since not much time was left for coding and testing a few intensive brainstorming sessions was made before it was agreed to store all route data in the form of a text file in a specific directory on the phone and then later on create a text handler on the website which would parse the content of an uploaded file and store it in the database.

This was accomplished by letting each route be represented in the form of a text file. Every time a new route is created a new text file with the route id and name as filename is also created in a special folder on the cell phone. If an existing route is

chosen which has been created in offline mode before then all new data will only be appended to the end of the textfile. All GPS data is then sent with the same syntax as it is sent to the server

Afterwards when the user has access to internet he only needs to login to his account on the website and upload the textfile which will create the route with all data in the database. If he has saved a message with an attached photo then the photo name and size, which also is stored in the textfile, is inserted into the database. Consequently he must manually upload the photo he has taken so it can be reflected on the website.

3.11 Adding functionality to the website

The system by now had become so multi facet that it could be extended in all kinds of ways. It was therefore important to put focus on the most important and needed features and to avoid drifting into smaller niches which might hinder further development. This was accomplished by yet again having a series of discussions with the involved parts at Svep to evaluate the progress so far and the main functionality which was still lacking in the system.

3.11.1 Social connections

Since the idea behind the entire project was to share routes with other users it is important to have the ability to establish a connection with other users.

First off a page dedicated for the user's personal profile was created. The profile would provide an overview of the user (using the user's own input) and his routes.

To keep it very simple only a box with personal text, an avatar and a list with the user's route was created along with general information such as when the user registered, how many routes he has and the like. The personal profile was kept simple just to prove that full scale community features can be implemented if this is wished when marketing the system.

Also a hierarchy access system, using the "Friends" table in the database, was added to the visibility of routes where the owner of a route can administer and grant access to those users who are allowed to view his route. If an unauthorized user tries to view his route nothing will be displayed. A user can also make his route public which enables non-registered users (guests) to view his route. This was enabled to strengthen the community feeling of the site so users can show their routes to each other even though they might not know each other and thus perhaps create a bond.

It should be noted that in the personal profile of the user those routes which the visitor does not have access to are not listed in the user's list of routes.

3.11.1.1 Friends

Once the personal profile and route access system was built the next step was to create and administer friendships with other users. Again this was made very simple to show that such capabilities exist for the system and can be further developed later on.

A user can request friendship with another user in the members' list, where all users are listed, as well as in the user's profile. A link was added in the members' list as the personal profile and if clicked would notify the other user. If the other user accepts the friendship a friendship relation is created between them in the database which is also reflected on the website and the MIDlet. On the website the friend is listed in the user's friends' list. Also when creating a new (website and MIDlet) or editing an existing route (website only) the user can grant access to the friend to view his route.

3.11.1.2 Groups

When implementing the support for friends the need for another feature aroused; support for groups. For example if a scouting team wishes to map their route, then all team members would want access to the route. Now instead of the creator of the route adding all team members as his friends and then grant them access to the route it would be much more natural to only grant access to a certain group and automatically all members of that group would have access to the route.

The group functionality was implemented in such a way that all users can create a group; they then become the owner of the group and administrate it. This was also kept simple just to prove what community capabilities the system has. All users can become members of the group and they then have access to the group's profile page where information about the group, the members and a list with all routes is shown. The user then also has the ability to grant access to the group when creating a new route (website and MIDlet) and editing an existing route (website).

3.11.2 Dashboard

As the website was growing larger, more user interaction and navigation was needed to reach commonly used functions. Viewing other and personal routes is the most essential function used by the user but in order to see if a friend has uploaded a new route the user must navigate to the friend's profile. Hence a personal dashboard was created which the user is taken to upon login. The dashboard provides a global overview to the user where the latest information is seen. This includes public routes, friend routes, group routes and personal routes. Also the user's friends and new friend requests are seen. The dashboard can further be developed if new functionality is added. For example if a comment system is implemented where users can comment on each other's routes then the dashboard could be extended to list the latest comments as well.

It could also be used to display messages from the administrators which needs to be broadcast to all members.

Coding the dashboard was relatively simple. A number of database inquiries were written to reflect the different sections and were then presented in lists.

3.12 Adding support for built in GPS

The next major step in the development was to add support to the MIDlet for cell phones with built in GPS.

Java has a very powerful and neat application user interface for developing GPS applications called “Location API”. First off a criterion is determined for selecting the location method. The criteria include fields such as response time, altitude, speed and accuracy. Once the MIDlet obtains a “LocationProvider” instance which meets the criteria, the object can obtain the location in two ways:

- Invoking a method synchronously to get a single location.
- Register a listener and get periodic updates at defined intervals.

The location data is then abstracted by the class “Location” which contains coordinates, time stamp indicating when the location measurements were taken as well as speed and textual address (if available).

There are two classes in which the coordinates are represented:

- A “Coordinates” object which contains the latitude and longitude in degrees and the altitude in meters of a point.
- A “QualifiedCoordinates” object which contains the latitude, longitude, altitude and further also an indication of their accuracy given in the radius and area.

Since the GPS tracking system will need to retrieve location data frequently and transmit it to the server it was a obvious choice to register a listener which, similarly to a thread, is invoked at the defined interval. When invoked the coordinates then can be extracted and passed on to the server.

The following segment of code illustrates how to create a “Criteria” object, define a characteristic for the criteria, get an instance of the criteria and then set a location listener class which is invoked periodically at a given interval:

```
// Set criteria for selecting a location provider:  
// accurate to 500 meters horizontally  
Criteria criteria = new Criteria();  
criteria.setHorizontalAccuracy(500);  
  
// Get an instance of the provider  
LocationProvider provider = LocationProvider.getInstance(criteria);
```

```

// Set a "LocationListener" class which is invoked by the
"LocationProvider"
// at the given interval.
// Parameters: (LocationListener, int interval,
//             int timeout, int maxAge)
provider.setLocationListener(gpsScreen, 1, -1, -1);

```

A class implementing the "LocationListener" instance can then be created where location data is acquired. Below is a small code snippet of how the fetching of GPS data periodically using a "LocationListener" was implemented:

```

public class GpsScreen implements LocationListener
{
    public void locationUpdated (LocationProvider provider,
                                Location location)
    {
        // Throw out invalid location updates.
        if ( location.isValid() )
        {
            // Update the pedometer data.
            QualifiedCoordinates coordinates =
                location.getQualifiedCoordinates();

            if ( lastCoordinates == null )
            {
                // Just starting.
                lastCoordinates = coordinates;
                startTime = System.currentTimeMillis();
            }
            else
            {
                // Record another position.
                totalDistance += lastCoordinates.distance(coordinates);
                float averageSpeed = totalDistance /
                    (System.currentTimeMillis() - startTime) * 1000;
                lastCoordinates = coordinates;
                averageSpeed = convertMPStoKMH( averageSpeed );
            }

            // Update the location data.
            double lat = coordinates.getLatitude();
            double lng = coordinates.getLongitude();
            double alt = coordinates.getAltitude();
            float s = location.getSpeed();
            s = convertMPStoKMH( s );
            float c = location.getCourse();
            long time = location.getTimestamp();
        }
    }
}

```

Once all relevant GPS data was retrieved it could easily be sent on to the server and stored in the database.

3.13 Marketing the system

Building an extensive and powerful system using the latest technology is indeed very inspiring but finding an innovative way to market and profit from the system is also an exciting challenge. GPS based applications are truly the future and there are many possible areas of use. The built GPS tracking system has all the relevant GPS data it is just a matter of how to use it. Thus four possible customer groups have been presented below which might be interested in the system.

3.13.1 Logistic

Among logistic companies there are already a couple of tracking systems prevalent. These are however rather large and expensive.

On the other hand the built GPS tracking system offers tracking of objects for a minimal fee using only a cell phone which is very competitive. This could be very appealing to smaller logistic companies or other companies and organisations which need to keep a track on their moving objects. Apart from trucks, cabs, emergency services and the like a whole new group of customers might emerge which would be interested in adding tracking to their activities due to the minimal requirements needed to run the GPS tracking system.

The system could be sold with one start up sum and then a smaller fee per account.

3.13.2 Sport

The GPS tracking system could also be used as a sport application. The maps with drawn routes would then serve the function of a training journal. Based upon the time, length and speed interesting calculations and graphs could be presented to the user. This could also be extended with affecting factors such as weather conditions, terrain and altitude when training. Since all that is required from the user is a cell phone with a GPS device basically everyone interested can get started immediately without needing to buy extra equipment.

To the sport application a social dimension could also be added where users get to know other users who perhaps train in the same area (based on the GPS location) or has similar training and body condition. A competitive edge could also be provided where users earn points through their training through different categories, such as individual progress (where the system calculates how the user's body condition has improved over a period based upon how better training values he has got), peak training, endurance training or the like. The highest ranked users could then be rewarded with gifts from sponsors and extra community fame on the website.

The GPS tracking system can also be used in a more ideal sense where it sponsors local sport events. For example the contenders of the annual Stockholm Marathon or “Vasa loppet” can be followed live on the website using the GPS tracking system. This would generate good credit for the company. Also profits could be made by having advertisement on the website which shows the route as well as the contenders through a very small fee can be given the opportunity to save their route on the server.

3.13.3 Travelling

Another field of application, which also was thought of when the project was only an idea, is a travelling application. Current travelling applications on the internet provide the ability to post text, photos and sometimes video of the user’s journey. Users can read each other’s travelling diaries and be inspired with new destinations. This also produces a good deal of advertisement which is the main source of revenue.

With the GPS tracking system a whole new dimension can be provided. Instead of mere text and photo the photos will be marked on a map with the actual position of where they were shot along with relevant comments. The user can illustrate his journey on a map which depicts the terrain and environment. In such a way other users will have clearer picture of how the journey was, where the photos were shot and the scene experienced.

If the GPS tracking system is used in a travelling application it might not directly earn income by itself but it will enrich the content and provide a unique functionality to the application which will give it a huge leap over its contestants. Hence it will attract more users and the company earns more money indirectly through the GPS tracking system.

3.13.4 Community

The best and most innovative implementation of the GPS tracking system would probably be in a social community, preferably integration into an already established community. The system would provide a whole new way for users to interact with each other based upon their current location. It would also enable location targeted advertisement.

Tracking a route is interesting when travelling, working out, making field trips and the like but is not so significant on an ordinary day. Rather what is relevant is where my current location is and more importantly; is there an ongoing event where I am which might attract other users as well?

If I go to this event, meeting, party or the like I can update my current location as well as send photos from the place to the server. Other people and friends can then through

the community view what hotspots there are in town, what is happening where as well as see where other appealing users are currently.

Further through a web interface the meetings places can send out offers to users either resident in the local town or already on spot.

Imagine the following scenario. A Friday evening the user “Catwoman” notifies that she is at “Coffee House”. She and other users there can upload photos to the community which is presented as a live photo stream online. “Trinity” sees through the photo stream that her friend “Catwoman” is at “Coffee House” and the place is rocking so she also goes there.

Based on the given location the users know who else is there. They can then contact each other through SMS-like messages. They could also through a minimized cell phone version of the community view other users’ profiles and see if they have anything in common before making contact. This would in other words enable a whole new form of social contact which isn’t limited to the computer screen. The communication would be of two sorts: user to user at “Coffee House” and online user at the community to cell phone user at the “Coffee House” (and vice versa).

A third form of communication is also enabled which generates the profits: through the web interface “Coffee House” can send out offers to users in place as well as outsiders. Through the GPS data the offers and advertisement is restricted to users in the local town.

Examples of communication:

“Coffee House”:

- “Free entrance fee for all born in November!”
- “Buy two coffees and get a free cake!”

Activities can also be arranged in this way:

“Coffee House”: “All brunettes come to the sofa for charades!”

The community: “All 20 year olds gather down town for collective bowling!”

Also, as a means of giving benefit for the users of the GPS system, special prices and offers can be given to those who show, for example with their cell phone, that they are using the GPS system.

This technique doesn’t necessarily require a GPS enabled cell phone from the user. For example those who don’t have a GPS a list of popular predefined places can be presented in the cell phone which the user can select from. Thus each time the user sends photos or messages they will be attached to the predefined coordinates.

Accordingly the profits will be generated in a number of ways:

- 1) For each advertisement and offer sent out by “Coffee House” the community gets revenue.
- 2) “Coffee House” can attract more customers through advertisement on the community. This can be accomplished in a couple of ways such as, having their logo marked on the map which presents and overview of all hotspots in town or be marked somehow in the predefined list of hotspots for those who don’t have a GPS in their cell phone.
- 3) Both “Coffee House” and the community will get a greater and at the moment unique content value where the users can contact each other based on where they currently are located.

Such an implementation would truly revolutionize the social experience where the virtual life is merged together with the real life in a way that is not scaring but rather appealing and gives the users the opportunity to get in contact with other persons whom they might never have spoken with before!

4 Description of the system

This section describes how the GPS tracking system works.

4.1 The MIDlet

The MIDlet is primarily used for sending coordinates. The coordinates are collected under a specific route and then presented on the map.

The user starts by connecting to the server and then logs in. He has the option of either creating a new route or continuing an existing one.

If he creates a new route he is brought to a screen where he can enter route specific information, such as name, description and who is allowed to watch the route (i.e. only for private use, for the public, for all his friends, for selected friend, for all groups he is a member of or only for selected groups etc).

After the route is created he is brought to a screen where he sets the interval of how often coordinates should be sent. If he chooses the option of existing routes all his uncompleted routes are fetched from the database and presented in a list. He can then proceed by choosing the route he wants and is also brought to the interval screen.

After choosing an interval the MIDlet scans the cell phone for GPS devices. Firstly the serial port is checked to see if a HGE-100 GPS device is connected. If not the MIDlet goes on to see if the phone has a built in GPS device.

Once a GPS device is found the user will be presented with a screen showing relevant interesting location data. He now has the option of starting and stopping the transmission of coordinates to the database. He can also send messages or take photos and send along with the coordinates. The messages and photos are later presented on the map on the website on the coordinates they were sent from.

The data which is sent to the server is received by another Java application which listens to incoming connection requests and launches a server handler for each request. The server handler is like a conveyor between the MIDlet and the database. Data is fetched from the database and sent to the MIDlet when the MIDlet requests so. Data is also sent from the MIDlet to the server handler which forwards it to the database for storage.

All routes have a unique id and the coordinates are stored under a specific route in the database. Thus when a route is chosen on the website only coordinates with the specific route id are presented on the map.

4.2 The website

The website is meant as an administration, a tool and a presentation of all routes. There is a social aspect of it where the user has his personal profile and friend list enabling him to communicate with other users; however the main focus is on the actual mapping of routes.

The user firstly logs in to the website and is then presented with an overview of the latest events, such as new routes by his friends and groups he is a member of, new members, friendship requests and the like.

The user can create a new route; the structure is then very similar to the MIDlet. He can enter a name, description and grant access to those he wishes. He can also edit existing routes and change the before given information. Further he has the option of setting his route as completed; it will then not be fetched in the MIDlet when he wants to continue sending coordinates to an existing route. Another useful function is the ability to upload photos which are taken with a digital camera. By scanning the EXIF data in the taken photos, which contain a timestamp of when the photo was taken, all coordinates of a specific route are checked and the photo will be placed on the coordinate with a timestamp closest to the photo's timestamp. This enables the user to take high quality photos with their digital cameras while being on a roundtrip without needing to bother with the MIDlet.

When it comes to socializing, the user has a personal profile and can get in touch with other users and groups. He can start a friendship with other users. He can also create or join groups. The group feature is of great use when it comes to route sharing. Imagine this scenario: a scout team wish to share their routes with each other. If the group feature is not available a team member must add all other members to his friends list and each time he creates a new route he must grant access to all these friends. If a person enters or leaves the team he must either add or delete that person from his list as well. With the group feature instead the scout team creates a group called for example "Helsingborg Scout Team". When each member creates a new route he only grants access to the group "Helsingborg Scout Team" to watch the route. Now automatically all members of the group can watch the route and would a person enter or leave the group then he has just been granted or lost the access to the routes.

The routes themselves are presented on a map with a number of functions. A line is drawn through the coordinates to better visualize the route. If messages are sent these are marked on their specific coordinates with a message icon. If photos are taken these are marked on their specific coordinates with a camera icon. There is the possibility to change the map view to a satellite view which comes in handy when a route is in an area without roads, since the satellite view will then visualize the terrain and environment.

Apart from the map presentation there are several interesting facts which can be presented along with the route. These include among other the length and duration of

a route, the average speed throughout the route, the distance between different messages and the like. With a bit of coding and calculating even more facts could be added which are interesting for different groups. For example when it comes to athletes an estimation of how many calories have been burnt on the way could be added.

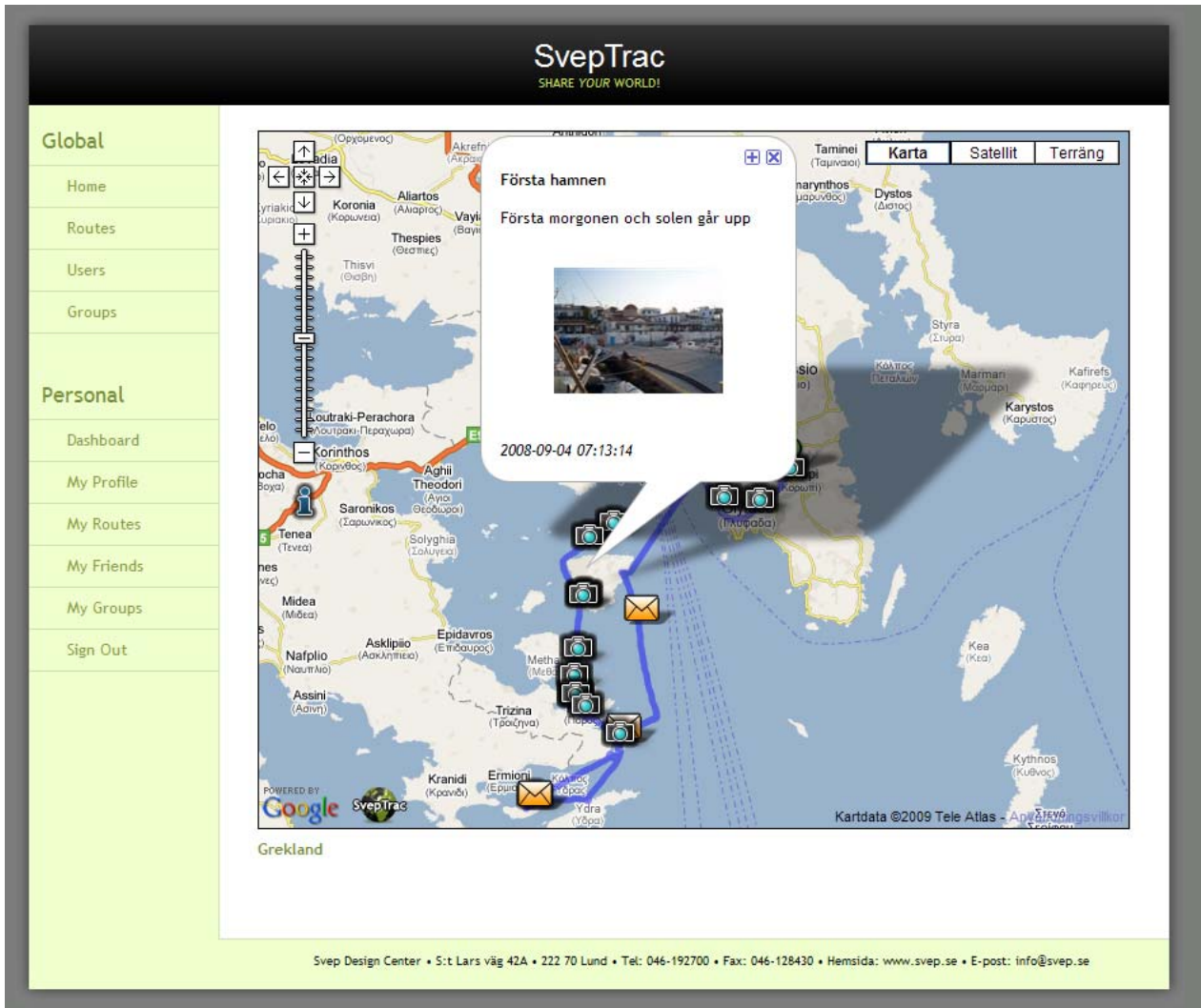


Figure 8: Shows the end result of the website where a sailing route through Greece is presented. The blue line is the tracking line and marks how the sailing was done. The camera icons all contain photos, which can be displayed in full resolution while the postcard icons only have text messages.

5 Result

Produced were a database, an application for the cell phone (MIDlet) and a web application (website) presenting the GPS data. The database was used for storage while the MIDlet communicated with a Java application on the server which was connected to the database. The website fetched the data from the database and presented it on a Google based map (Google Maps).

The objective of the project was to create the base of a system which stores and exchanges routes based upon GPS data. The system would consist of two main parts: a MIDlet and a website. Hence the objective of the project was accomplished.

The purpose of the project was to create a system which enables end users to share their routes and journeys with each other. The aim is to take social interaction into a higher level where ordinary text and multimedia is combined with visual maps of the user's personal routes and journeys. This was also accomplished by adding the social features to the system where only certain users or groups could view a route. If the system is marketed in the form of a community where location targeted advertisement is used (see chapter 3.13.4) this would also aid in taking the social interaction into a new dimension.

Further the problematic areas of the project involved minimizing the battery usage and data transmission costs. To find out the resource usage of the GPS system a test was made. The test was designed to track a route and then calculate the expenses. This was accomplished by driving car from Lund to Helsingborg, approximately 49 kilometres, while tracking the drive. The results where the following:

- Cell Phone: Sony Ericsson Z610 with a HGE-100 GPS device.
- Duration: 50 minutes where GPS data was sent to the server in a 30 second interval.
- Battery Usage: The battery dropped with 8%, this cannot be an entirely accurate value since the cell phone had a tendency to not show the correct battery level.
- Data Transmission: 77 coordinates where totally sent to the database. 29kb was transmitted during the entire test.
- Cost: The total cost was 0.77 Swedish kronor. That gives a charge of 0.057 Swedish kronor for each coordinate to send. The cell phone card used was a Comviq card where GPRS data costs 15kr/MB.

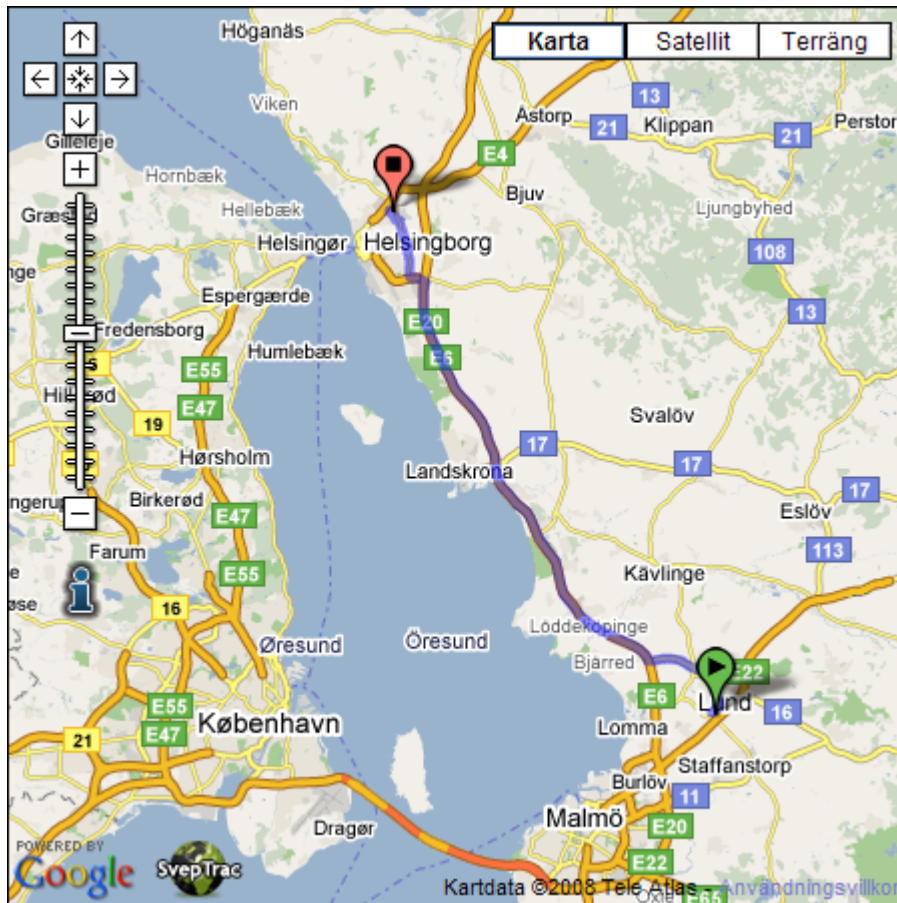


Figure 9: Shows the result of the test where the drive from Lund to Helsingborg was tracked in an effort to calculate the resource usage of the system

The test proved that the GPS tracking system indeed is a low resource consumer and hence can be used in a real life application without unappealing costs to the user.

To sum it all up the results reflect a successful project where the aim has been reached and the purpose has been accomplished.

6 Summary

As this was the first major commercial project done naturally a great deal of experience has been gained.

6.1 Difficulties

There were a number of difficulties on the way. First of considering it was such a large project it was hard to know where and how start as well as foresee future needs. As so many programming languages were involved it was also a bit inconvenient to shift between them constantly. Due to the many affecting factors troubleshooting in case of errors could be rather tricky at times.

When it comes to the GPS signal it was a bit weak which slowed down the development since no positioning was received at poorer weather conditions.

The main difficulties on the software side were primarily the Java bit. Since J2ME is a scaled down version of Java suited for cell phones it lacked support for some Java native functions which was needed during the development. The consequence of this was to develop personal functions adjusted after J2ME limits to do the same thing, which cost extra coding time.

The Google Maps coding on the other hand went quite smoothly since a lot of support was available online. However when errors arose it was a real pain to debug them since all code goes through Googles servers and are out of reach.

6.2 Improvements

When it comes to the time planning it was a bit complicated to set up a proper time plan since the outcome and goal of the project wasn't entirely clear. However smaller milestones could have been set and once a milestone was reached a new one could be set which reflected the need and vision of that stage.

Even though the goal wasn't clear it probably would have been better if an end goal was decided, after a series of discussions and brainstorming, including which customer group is targeted and what the system is meant to be used as. If a clear goal had been set it would ease the development greatly and unnecessary code wouldn't have been produced.

It would also have been smarter to select a couple of cell phones from different platforms which the system would work well on, rather than trying to provide

compatibility with all cell phones. Since full compatibility was aimed for extra time was put to make the system work on alternative cell phones then those used during the development and the end result was that the system lost some of its stability.

Another point worth mentioning is not to focus on flashy design but rather simple intuitive interfaces, which might be uglier, but function well. Colour and design should be added at the end of the project when it's time for marketing since it probably will change many times during the process otherwise and take unnecessary code time (which also was what happened).

6.3 Lessons learnt

There were a number of lessons learnt from this project, both project- and tech specific. One of these was how to plan a project from the very foundation and build a large system on it. Along with this the importance of regular discussions and brainstorming sessions was realized as well as the significance of having a constant futuristic vision.

Since several platforms were used another lesson learnt was the interaction of these platforms and how to code cleverly in order of them to work smoothly together.

A great deal of network communication was also learnt as well as how to set up a client-server network and establish a stable, effective and fast communication between them. What could have been done better was to create a personally specified network protocol which the communication was bound to follow. This would have made the communication far more reliable and increased the speed of the transmissions since the amount of data would be decreased. Further bit representations of the data could have been transferred instead of ASCII signs which would also increase the stability of the transmissions since there would be less risk of data corruption. The protocol could also be shaped in such a way to specify the length of an incoming data transmission, with a start and stop bit – and if this amount of data isn't received the server will keep listening until all data is received (or a timeout aborts the operation). But since ASCII signs were used instead it happened that when larger amounts of data were transferred, such as a photo, data would get corrupt resulting in an invalid photo.

When it comes to the tech part a great range of new platforms were learnt such as Microsoft's .NET including C#, MSSQL and Visual Studio, J2ME programming for the cell phone and JavaScript for advanced web development. Also the GPS protocol, Java's Location API and Google's API for their Google Maps application was learnt.

6.4 Last words

The journey has been long but now when looking back the view is astonishing! The lessons taught have been tremendous and it has been exciting to put into practise all the knowledge collected throughout the years of studies. It has been very paying to have shaped something from the ground and experiencing the process to a fully working system. Finally it has also been truly inspiring to work with the latest technology and having been given the opportunity to innovate a new future!

7 References

Google Maps API Documentation

<http://www.google.com/apis/maps/documentation/>

(April 2008)

Google Maps API Tutorial

<http://econym.googlepages.com/index.htm>

(April 2008)

J2ME Coding samples

<http://www.java2s.com/Code/Java/J2ME/CatalogJ2ME.htm>

(Mars 2008)

J2ME Tutorial, Part 2: User Interfaces with MIDP 2.0

<http://today.java.net/pub/a/today/2005/05/03/midletUI.html?page=1>

(February 2008)

MSDN: Microsoft Developer Network

<http://msdn.microsoft.com/en-us/default.aspx>

(April 2008)

NMEA 0183

<http://www.kh-gps.de/nmea-faq.htm>

(February 2008)

The Official Microsoft ASP.NET Site

<http://www.asp.net/>

(April 2008)