JoggerLogger
Bringing GPS and jogging together

Stig-Erland Hansen
Department of Computing Sciences
Østfold University College
Halden, Norway,
stig.e.hansen@hiof.no

June 2, 2006

Abstract

JoggerLogger is a project where the goal is to enhance the experience of jogging by using GPS and standard web technology. This has lead to a design consisting of three parts: a mobile client, a repository and a web client. The mobile client, which consists of a mobile phone and a GPS receiver, can be used by joggers to record routes and results. These data can be sent over the Internet from the mobile client to the repository in order to share them with others. The information in the repository can be managed and viewed using the web client.

This design has been realized in a proof-of-concept implementation to show that such a system is feasible to develop. The mobile client is implemented using J2ME, and the repository and the web client is implemented using J2EE.

Keywords jogging, GPS, Java, J2EE, J2ME, mobile phones, Hibernate, Spring

Contents

Abstract **i**

1 Introduction **1**

2 Background **3**
2.1 Related work . . . 3
2.1.1 FRWD . . . 3
2.1.2 Trac Trac . . . 4
2.1.3 Sportsim . . . 4
2.2 Programming methodology . . . 4
2.3 Technologies . . . 5
2.3.1 Mobile phones . . . 5
2.3.2 GPS . . . 6
2.3.3 WMS . . . 6
2.3.4 Google maps . . . 6

3 Design **8**
3.1 Scenarios . . . 8
3.1.1 Create a route . . . 8
3.1.2 Find a route and run it . . . 9
3.1.3 Virtual race . . . 10
3.2 Repository . . . 10
3.3 Mobile client . . . 11
3.3.1 Create a new route . . . 11
3.3.2 Run a route . . . 11
3.3.3 Run a route against a virtual opponent . . . 12
3.3.4 Communication with the repository . . . 12
3.4 Web client . . . 13
3.4.1 Account management . . . 13
3.4.2 Find routes . . . 13
3.4.3 Replay runs . . . 14

3.5 Additional features 14
3.5.1 Training plans 14
3.5.2 Automatic route creation 14
3.5.3 Tagging 15
3.5.4 Rating 15
3.6 Applications 15

4 Implementation **17**
4.1 Mobile client 17
4.1.1 Technological considerations 18
4.1.2 User interaction 19
4.1.3 Details 26
4.1.4 Improvements 30
4.2 Repository 31
4.2.1 Technological considerations 31
4.2.2 User interaction 32
4.2.3 Details 32
4.2.4 Improvements 37
4.3 Web client 37
4.3.1 User interaction 37
4.4 Summary 40

5 Testing **42**
5.1 Test objectives 43
5.2 Test group 43
5.3 Test procedure and tasks 43
5.3.1 Create a route 44
5.3.2 Run a route 45
5.3.3 Race against an opponent 45
5.4 Test results 45
5.5 Improvements 46

6 Conclusion **48**

List of figures **52**

List of tables **53**

Chapter 1

Introduction

Jogging is a simple exercise form for people of all ages in all countries. No equipment is needed since it is as simple as moving from one place to another by using one's legs. This simplicity can result in jogging being monotonous and boring. Still, there are countermeasures that can be applied to prevent this, and in the writer's opinion, there are especially two countermeasures which are important.

First, having some form of competition can make the jogging experience better by motivating the jogger to make an extra effort. One of the most common competition methods is to jog against someone else, and this method can be highly effective in motivating joggers. However, in order for it to be effective, the joggers should have similar fitness levels. If there are great disparities in the fitness levels, it is likely that either the joggers end up running by themselves or one of them jogging at slower pace than he or she is capable of. Another problem with this competition method is that the jogger must find someone to compete against which is not possible for everybody.

If the jogger does not have somebody to jog against, he or she can use another common competition method of jogging against oneself by timing each run. This method is effective for comparing results after finishing a run, but the end time is hard to use during a race to find out how the current performance compares with previous runs. In conclusion, the two competition methods discussed in this paragraph and the previous, which also probably are the two most common competition methods, have problems that can make them hard or impossible to use.

Second, varying the routes that are jogged can make it more exciting than jogging the same old routes. In addition, varying routes may be necessary because of uncontrollable circumstances that makes one or more routes inaccessible. For example, it is not a pleasant experience to run in the woods

if it rains heavily. In this kind of weather, it is probably better to run in a different terrain. However, the problem is to have good routes to choose between.

Both of these two countermeasures have their set of problems as described above, but all of them can be solved by creating a worldwide, publicly available online repository of jogging routes and results. In this way, joggers can share their routes and results with others, and get access to a wide range of routes to run and results to compete against. This paper will present this system in detail and how it can be implemented using Global Positioning System (GPS) receivers, mobile phones and standard web technology. The working title of this system is JoggerLogger.

This paper is divided into 5 chapters. Chapter 1 is this chapter which gives an introduction to jogging and problems with it. Chapter 2 gives an overview of related work, the programming methodology and the different technologies used in this project. Chapter 3 details the design of the system, while chapter 4 describes how the different parts of the design were actually implemented as a proof of concept. Chapter 5 explains how the testing of the implementation was performed and which discoveries that were made. The paper ends with chapter 6, the conclusion.

Chapter 2

Background

In this chapter, the necessary background information for this project will be given by first introducing similar and related work, second by outlining the programming methodology used, and third by giving a short description of all the different technologies that were utilized.

2.1 Related work

A lot of work has already been done in the area of GPS and jogging which has resulted in commercial products. Three of these products will be presented below. Interestingly, they have sightly different focuses and concentrate on different aspects of jogging.

2.1.1 FRWD

FRWD[1] is a GPS recorder that records the following variables of a run: route, location, speed, distance, heart rate, altitude, temperature and air pressure. The information can be viewed either on a mobile phone or on a computer by installing software delivered by FRWD Technologies Ltd. The software for the mobile phone enables the user to view the performance information from FRWD during a run. For example, it is possible for the user to view his or her current heart rate and speed. The software for the computer, on the other hand, makes it possible to replay and analyze the performance of one or several ended runs. In addition, the user can share his or her results with others by sending an email.

To summarize, this product supplies a tool for joggers to evaluate their own performance during and after a run has ended. However, the notation of sharing routes through a central repository is not the focus of this product.

2.1.2 Trac Trac

Trac trac[2] is a system for logging the movement of participants of activities like team building or a sports event. This is done by equipping each participant with a light-weight combined GPS and GPRS receiver that sends its position to a server. The position of each participant can be viewed either in real time or after the activity has been completed through a Java applet in a web browser.

This system is targeted towards big sports arrangements and corporations that do team building, and it is not for personal use. It has already been used in big events like the Berlin marathon and Nordic Orienteering Championships.

2.1.3 Sportsim

Sportsim[3] is a little different than the other systems described above since it does not deliver any performance information during a run to either the jogger or other people that are interested. In fact, the system is not concerned with how the actual logging of the movement is done, and the user is free to choose the GPS unit as long as it has the ability to log movement and transfer the log to the computer. Instead, the focus of the system is to interpret these logs and replay them on a computer while displaying information about the performance like speed and distance.

In addition, the system enables the user to create shared races where everybody can join as long as they have physical access to the route of the race. The shared race can either be an actual race where the participants start at the same time, or it can be a virtual race where each participant runs whenever it is most convenient.

2.2 Programming methodology

The programming methodology used in this project is influenced by eXtreme programming (XP), but it is does not conform to all of its rules. Some of the rules are impossible to follow since this is a one man project, while XP is a programming methodology designed to be used by teams of programmers. For example, the rule of programming in pairs is impossible to adhere to when there is only one person programming.

Other rules are impractical to follow to the extent specified by XP. For instance, the rule of test-driven development and the principle of writing tests before any code is written will probably take time to adjust to, although it is a great rule. As a result, a more gentle approach will be taken where

some tests are written for the code either before or after the code is actually written. In this way, it is possible to become familiar with tests and follow XP more strictly on other projects.

Nonetheless, the rules of the planning game, simple design and small releases are followed more strictly. The planning of the project follows the planning game rule in both release planning and iteration planning. In release planning, a plan of the features that should be developed for a given release will be made where the features are ordered according to the cost and importance. However, the plan is not constant in any way and may change during release development.

Many small releases will be developed to avoid setting too high goals that are not realizable within the time limit of the project. In addition, the design will be kept simple and according to current requirements instead of focusing on future requirements which might not be implemented anyway. During the development of a release, iteration planning will be performed together with the supervisor for the project at weekly meetings where the progress of the project is evaluated and the course for next week is laid out.

2.3 Technologies

2.3.1 Mobile phones

The mobile phone as a device has grown tremendously since its initial release. In the beginning, the mobile phone adhered to the name of being a device that allows people to talk to other people while being on the move. However, this is not true any more, and most mobile phones today include many other features like sending text messages using either Simple Message Service (SMS) or Multimedia Messaging Service (MMS), browsing the Internet, planning the day using the internal calendar, taking pictures and listening to music. It is even possible for third-party companies to develop applications for them.

Furthermore, the adoption of mobile phones has grown tremendously since the initial release, and it is today widely used "[i]n most of Europe, wealthier parts of Asia, Africa, the Caribbean, Latin America, Australia, Canada, and the United States ..." [4]. Thus, applications developed for mobile phones can reach a lot of people which means that it is possible to make a lot of money.

```
http://www2.demis.nl/mapserver/request.asp?Service=WMS&
    Version=1.1.0&Request=GetMap&
    Layers=Countries,Borders,Coastlines&Format=image/gif&
    BBox=0,55,40,75&SRS=EPSG:4326&Width=400&Height=400
```

Figure 2.1: An example of a request to a WMS using the GetMap operation.

2.3.2 GPS

Global positioning System (GPS) is a satellite navigation system originally built by the United States Department of Defense. It consists of 24 satellites which orbit the earth and transmit radio signals about their current position [5]. This information is used by GPS receivers to pin point their position on the earth by using a technique called trilateration. In order to use trilateration, the GPS receiver must receive signals from four different satellites. GPS currently offers a accuracy of 4-20 meters, but there other techniques which can be used to increase this accuracy like DGPS [6].

2.3.3 WMS

Web map service (WMS) is a service which allows images of maps to be fetched by issuing requests over HTTP[7]. A WMS must support two operations: GetCapabilties and GetMap. GetCapabilties is used to get detailed information about a specific WMS such as the different layers it supports. GetMap, on the other hand, is used to download a image of a map. This image can be controlled to a high degree by the parameters given to getMap operation. These parameters include information like the reference system, the layers to include, the size and format of the image and the size of the map. An example of a GetMap request can be seen in figure 2.1.

2.3.4 Google maps

Google Maps [8] is a service which makes it possible embed interactive maps in a web page. The interaction with the map is done by dragging and clicking on the map, and interaction is similar to what is possible in a desktop application. The map can be included on any web page by getting a key from Google, which currently is free of charge, and adding some HTML and Javascript to the web page.

In comparison to WMS, Google Maps is a higher level service. It provides a user interface, but it does not allow the same detailed control over the

information that is displayed on the maps as WMS does. In addition, Google Maps is designed to be included in Web pages, where WMS can be used in other places as well.

Chapter 3

Design

JoggerLogger consists of four parts: the repository, the mobile client, the web client and a map service of some kind. The repository includes all the information about the users, routes and results. The mobile client is used by the jogger to record new routes and results and upload this data to the repository. The web client makes it possible for joggers to find new routes and view the routes and results that they have added. The map service provides both the mobile and the web client with map data. However, the map service is not directly a part of JoggerLogger, and it will be provided by an external source; Thus, it will not be further explained in this chapter. Figure 3.1 illustrates how the different parts relate to each other.

3.1 Scenarios

Three scenarios have been created for this project which describe how the different parts of the system are connected to each other and how it can be used by one or more joggers. The first scenario starts by describing how the creation of routes and results is performed by the jogger. The other two scenarios describes how this information can be used by others.

3.1.1 Create a route

The sun is shining and John is in the mood for jogging his favorite route. He puts on his training gear, equips himself with his GPS receiver and mobile phone with JoggerLogger (mobile client), and goes to the start location of the route.

Since no one has recorded this route before, he decides to create a new route. He signals to the mobile client that he is about to start jogging, and

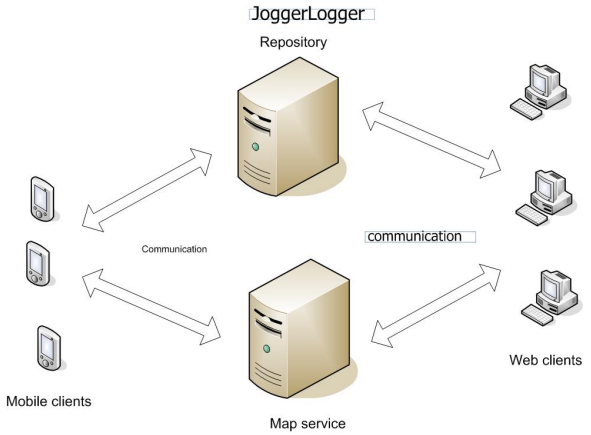


Figure 3.1: Overview of the three parts which JoggerLogger consists of.

the mobile client starts to log John's movement.

John pays no attention to the mobile client until he finally reaches the end of the route. He picks up the mobile phone and signals that he has reached the end. The mobile client responds by displaying information about the run. After viewing this information for a moment, John decides to store the route and result on the device. In addition, he decides to share his route and result with others through joggerlogger.org.

3.1.2 Find a route and run it

Mick wants to jog, but he is fed up jogging the same routes all the time and wants to try something new. Thus, he gets in front of his computer and visits joggerlogger.org where he logs on to his account. He starts browsing the different routes which other joggers have added. The browsing is performed by moving around on a map where all the start locations of the different routes are placed.

After some browsing, Mick finds the route John added previously and decides to view the details of it. An outline of the route along with information like the length of the route are displayed. This information makes Mick certain that this is a route he wants to jog. Therefore, he adds the route to his favorites on his account and logs off.

Then Mick picks up his mobile phone and choses to synchronize with his account on joggerlogger.org. The mobile client connects to joggerlogger.org and figures out that a new route has been added which must be downloaded. After the route has been downloaded, Mick choses to run this route. The mobile phone displays a map with Mick's current position and the start

location of the route. Mick uses this map to find his way to the start of the route.

Arriving at the start of the route, Mick starts jogging and signals this to the mobile phone. The jogging goes fine until one point in the route where the road split in two directions. Mick trusts his vague memory of the outline previously viewed online and runs in one of the directions. However, this direction is wrong. Since the mobile phone knows which direction he really should be headed, it signals to Mick that he is departing from the route. In response, Mick picks up his mobile phone to see which direction he really should be headed.

A similar signal is given to Mick, when he is at the end of the route. Mick stops and views his performance before saving the result on the mobile phone. Then Mick decides to synchronize with his online account in order to share his results with others and compete about the best result.

3.1.3 Virtual race

John logs on to joggerlogger.org to see if there have been some changes. He notices that Mick has jogged his route and that Mick's result is better than his own. This irritates John, and he decides he wants to beat Mick. Thus, he adds the result as an opponent to his account and downloads this result using his mobile phone.

Then John goes to the start location of the route, picks up his mobile phone, chooses the route he is going to run and that Mick is to be his opponent and starts jogging. At first, John runs slower than Mick, and he is notified of this by the mobile phone at the first check point. In response to the notification, John increases his pace which results in John being in front of Mick at the next checkpoint. This position is held by John for the rest of the run, and John is eager to transfer the result to joggerlogger.org in order to show Mick who is the best.

3.2 Repository

The three scenarios presented in the previous section showed that sharing jogging information is an essential part of the system. Sharing is possible because of the repository where all the information about users, routes and results are stored. Each user has an account where all the data about the user is stored. This includes demographic data like name, address, age, location as well as data like friends (other users of JoggerLogger), favorite routes, opponents, and routes and results that the user has created.

This information is publicly accessible using either the mobile client or the web client. Both clients will communicate with the server using HTTP by issuing a get or post request. However, the web client is more tightly coupled with the repository since it is the repository that deliverers the web pages that make up the web client.

To sum up, the repository's main actions consists of either storing or fetching data from a storage of some kind, most likely a database.

3.3 Mobile client

The mobile client needs two devices to function correctly. The first device is a mobile phone where the mobile client will run. The second device is a GPS receiver which will deliver location information to the mobile phone. Both devices must support bluetooth since this technology is used to connect them together. In the next sections, the main functionality of the mobile client will be outlined.

3.3.1 Create a new route

This task is the most basic of all the task in the mobile client, and it will be be chosen if the route, which the user is going to jog, has not already been created. Interaction with the client is simple. The jogger must signal to the client when the jogging starts, and the client will start logging the jogger's movement. When the jogger reaches the end of the route, he or she must notify the client of this, so the client can stop logging. After the run, the user is presented with details about the run like time used, distance and average speed. The user is also given the ability to save the route and result locally on the mobile phone as well as sharing this information by transferring it to the repository.

3.3.2 Run a route

This task is used by the user if the route that is going to be jogged already has been created. It is similar to creating a new route and follow the same steps needed by the user except that the user must specify which route to run. Since the mobile phone knows the route the user is jogging, it is able to help him or her in several ways.

First, the mobile client can help the user find the right location of the route if he or she never has jogged the route before. This is done by displaying a map on the mobile phone where both the position of the user and the outline

of the route is displayed. In addition, the mobile client can prevent the jogger from starting in the wrong area by only allowing the jogger to start if he or she is positioned in the start location of the route.

Second, the mobile client can help the user run in the right direction during a run. This is done, as described in the previous paragraph, by displaying map to the user with the outline of the route and the current position of the user. The only problem is that the user probably will be unwilling to look at the screen at all time during a run. Therefore, the mobile client will, in addition to displaying the map, signal to the user by vibrating if he or she starts to run in the wrong direction.

Third, the mobile client knows where the route ends and does not need the jogger to signal when the route is over. Instead the mobile client will signal to the jogger when the route has ended by vibrating. This vibration must be different in intensity and interval than the vibration used when the jogger is differing from the route; Thus, the user is able to tell them apart.

3.3.3 Run a route against a virtual opponent

The task of running a route against a virtual opponent has the goal of providing a good competition method for someone that has no one to run against. The task is similar to running a route except that an opponent must be chosen in addition to a route. The opponent can either be a previous result created by the jogger himself or someone else who has jogged the route.

The resulting information is used to give feedback to the user during the run about the performance compared to the performance of the virtual opponent. The feedback is given at different check points during the run by vibrating. Different vibration will be used depending on the position of the jogger relative to the opponent. For instance, a short vibration is given if the jogger is in front and longer vibration is used if the jogger is lacking behind. If this information is not detailed enough, the user can view the more detailed information presented on the mobile phone.

3.3.4 Communication with the repository

The mobile client and the main repository need to exchange data. The mobile client has routes and results that the user wants to transfer to the repository, while the repository has routes and results that the user has marked as interesting that should be transfered to the mobile client. This transfer can be carried out in several different ways, and two possibilities have been discussed in this project.

The first method is to transfer the information indirectly by going through an extra device along the way in addition to the repository and mobile client. For instance, if a route should be transferred to the repository, it will first be transferred from the mobile client to a computer and then from the computer to the repository using the web client. This method is cumbersome and requires several steps by the user.

The second method is to allow the mobile client and the repository to communicate directly. This method is much simpler than the first method, but it may be more expensive since the transfer must be done over GPRS.

A real life implementation of the system could include both of these methods, enabling users to choose the one that suits them best. However, in this project, only the last method is used since this is the most interesting of the two.

3.4 Web client

The web client is the part of JoggerLogger that is used by the jogger to either evaluate past performance or to plan future runs. This functionality could have been put into the mobile client as well, but there are two reasons for not having done so. First, the mobile phone has limitations in both screen size, transfer speed and transfer cost which makes it hard to implement these features in a way that gives a good end user experience. Second, planning new runs and reviewing past results are probably done at home in front of a computer instead of on the move using a mobile phone.

3.4.1 Account management

The web client enables users to manage their accounts. Personal information like name, address and age can be changed. Furthermore, other users of JoggerLogger can be added as friends which the user then can challenge and receive notification if they add new routes and results. Moreover, favorite routes and opponents can be added to the account as well as viewing detailed information about them.

3.4.2 Find routes

The most important task of the web client is probably to help users find new routes. The different routes are positioned on a map which the user can explore and easily find new routes in the proximity of where he or she lives.

The routes that are interesting can be added to the user's account and then later downloaded using the mobile client.

3.4.3 Replay runs

Replaying runs is interesting for joggers who want to evaluate and compare previous runs with each other. Before starting replaying, the route and runs which should be replayed must be chosen. The web client will then present a map over the area of the route, and draw the outline of the route and the position of the jogger from the different runs on the map. During the replay, the position of the joggers will be updated, and performance information, like current speed and number of kilometers left, for each one will be presented.

3.5 Additional features

The description given of the three different parts of the system in the previous section is the core functionality of the system. However, there are many other features that could be added to the system as well, but including those in description would probably make it bigger and harder to follow. In addition, many of these features are highly experimental and might not be possible to put into practice. Therefore, a description of some of these features are included in this section instead, where they can be briefly presented without giving a deep description about their impact on the system as a whole.

3.5.1 Training plans

The web client could be extended to give the user the ability to create training plans. These training plans would typically include which routes and which opponents the user wants to run against at different times. In addition, the mobile client could be extended so that the training plans could be downloaded and added to the internal calendar on the mobile phone.

3.5.2 Automatic route creation

The system could use the information about all the routes in the repository to create new routes. This could be done by connecting different parts of different routes that intersect with each other. In this way, more interesting routes could have been made available for joggers to choose between.

3.5.3 Tagging

The routes could be made taggable just like links to web pages are taggable on del.cio.us [9] or pictures are taggable on flickr.com [10]. In this way, it can be easy for users to find routes with certain types of attributes. However, the tagging could be drawn even further by enabling tagging of portions of a route. This information could be used by the automatic creation feature described in the previous section to create highly customized routes. For instance, the user specifies the start position, distance and/or other attributes for the route which he or she will run, and the system creates a route that best fits these data.

3.5.4 Rating

The user could be given the ability to rate routes in order to make it easier for users to find the best routes. This feature is fairly standard and just about any system containing huge amount of user generated content today has it.

3.6 Applications

Although this project is targeted at joggers, it can be used in any sport which has the notion of routes. For example, cycling routes or cross-country skiing routes can be logged in the same way as jogging routes are logged. However, if this is done without modifications it will become difficult to separate jogging routes from other routes.

Another usage is to log the route to a specific place and get others to use this route to locate that place. For example, a person is throwing a party at a remote location like a cabin out in the middle of woods which it is hard to find for people unfamiliar with the surroundings. The person who throws the party can then log the route to his or her cabin from a place which is easy to find for almost everybody, and share it through the repository. Other people who will be attending the party can download this route, and use the mobile client to direct oneself in the right direction.

Generally, the findings discovered in this project can be useful for other systems as well. There are especially discoveries made in two areas that probably will be of interest to others. Firstly, discoveries concerned with logging the movement and creating routes from GPS tracks of users. How often should the position be logged, how can the log be used to create a route, and how can this route be used to direct others. Secondly, the discoveries made when trying to build up the repository of tracks through a community.

What is important when trying to build a community, how to attract people and how to keep them attracted to a product.

Chapter 4

Implementation

The implementation of JoggerLogger which will be described in this chapter is not full implementation that incorporates all the functionality mentioned in the previous design chapter. A full implementation of JoggerLogger was strictly not possible within the time limit for this project. Instead, the implementation will merely be a proof of concept where the goal is to show that it is possible to implement the full system.

A decision was therefore made to focus on the main part of the system, the mobile client, since the other parts rely on it. The repository rely on the mobile client to deliver routes and results, and the web client rely on information in the repository. Still, a light-weight implementation of the repository and the web client were made in order to show that it is possible to get the three parts to communicate with each other.

4.1 Mobile client

The implementation of the mobile client includes the main functionality proposed in the design chapter. It supports creating a route, running a route, racing against an opponent and uploading the routes and results created to the repository. However, the implementation has still some rough edges and the error control needs improvements before it can be ready for commercial use.

In addition, the mobile client communicates with a map service and displays the map images downloaded to the user. The map service that is currently used is a WMS from Statkart since WMS allows full control over the information that is downloaded.

The implementation of the mobile client will be presented in the next sections. First, it will outline the technological considerations made prior to

the actual implementation. Second, it will present the graphical interface (GUI) and the actions needed by the user. Third, it will explain in detail how the core parts of the mobile client are implemented. Fourth and lastly, it will describe which parts of the mobile client that have limitations and how them can be improved.

4.1.1 Technological considerations

The initial requirement of the mobile client was that it should run on a mobile phone and be able to communicate with a GPS receiver through bluetooth. This requirement was relatively wide, and there were a numerous of platforms available that it would be possible to use. Among these platforms, there were three that stood out, Opera platform, Python for series 60 and J2ME. Each of them were evaluated using the following criteria: functionality, ease of use, my knowledge of the platform, and how well the platform is supported by different devices.

The Opera platform is built around the mobile version of the Opera Internet browser, and enables developers to write applications using HTML and Javascript. This platform looks promising. It supports a wide range of mobile phones, it seems ideal for rapid development and the use of web as development platform make it possible to leverage knowledge acquired using the Opera platform when creating ordinary web pages. Nonetheless, the platform has an Achilles heel, which is the limited functionality.

The most important limitation is that platform does not support bluetooth communication, making it impossible for the client to directly communicate with a GPS receiver. Nevertheless, it is possible to circumvent this limitation, as done in the project MoBuddy [11], by having another program running in the background communicating with a GPS and sending this information to a server which the client later fetches the information from. This, however, will result in a delay before the client receives the position, which in this project would make the user experience poorer.

Python for series 60 is exactly what the name suggests: it is a platform that runs on Series 60 mobile phones that enables developers to write applications using the programming language Python. The platform support the functionality needed for this project, and it seems to be effective for rapid application development considering the dynamic features of Python a long with its practical and easy to use Application Programming Interfaces (APIs). However, the platform is only supported by series 60 phones, which are mostly Nokia phones. In addition, I have never used the platform before, although I have used Python on several projects.

J2ME is a acronym for Java 2 Platform, Micro Edition, which is basically

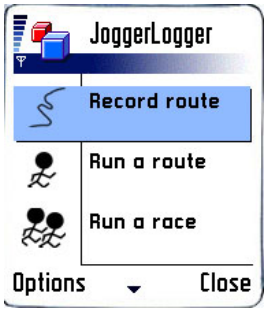


Figure 4.1: the main screen of the mobile client.

a scaled down version of the standard Java platform specifically targeted at small devices like mobile phones. The platform follows the same principle as the other versions of the Java platform does, which is "write once, run everywhere". In practice, it is probably more like "write once, test everywhere", but regardless of this the platform has gotten wide adoption by mobile manufacturers and just about every mobile phone created today has support for it. In addition, the platform supports the functionality needed for this project, and it has wide tool support, making development easy. Lastly, Java is my language of choice, and I have already developed small applications for J2ME, so I am fairly familiar with the platform already.

In conclusion, J2ME was chosen as the platform for implementing the client for its wide adoption in mobile phones as well as I am more proficient in the Java language and its tools.

The device that was used in development was a Nokia N70 mobile phone, and the mobile client has only been tested on this device. However, in theory, the mobile client should run on any device which support CLDC 1.1, MIDP 2.0, and JSR 82 (Bluetooth API) and JSR 75(File and PIM API).

4.1.2 User interaction

The mobile client, as most mobile application, have a main screen that enables the user to navigate in the application. This screen is presented in figure 4.1 where each task is represented as an item in a list. The list consists of the following five tasks: create a route, run a route, race against an opponent, synchronize and settings. Each of these tasks will be discussed in the next sections.

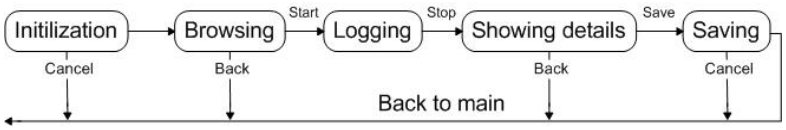


Figure 4.2: The five steps that are needed to create a route.

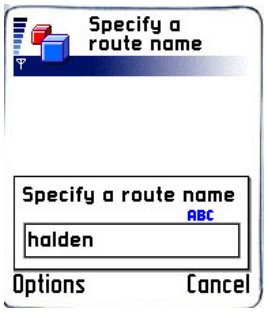


Figure 4.3: The initialization step for the task of creating a route.

Create a route

The task of creating a route consists of five steps, as showed in figure 4.2, which the user needs to do. These steps are similar, but not identical to the steps needed for the tasks of running a route and racing against an opponent. Therefore, each of the steps will be explained in detail in this section while only the differences between the steps are highlighted for the other two tasks.

Initialization In the initialization step, the user needs to specify the name for the route that will be recorded. The user is presented with a text box as showed in figure 4.3. This step could actually be moved later in the interaction model for this task, but the other two tasks need to perform initialization at this point. As a result, the initialization step is performed in the same place to keep the interaction model the same, hopefully making it easier for the user to use the application.

Browsing In the browsing step, the user is presented with a map over the area where he or she currently is located and he or she is represented as a circle on this map. This map, which is presented in figure 4.4, can either be panned or zoomed. Panning is performed by using the directional keys on the mobile phone and enables the user to see new areas of the map that currently are not visible. Zooming enables users to either increase or decrease the level of detail on the map. This is performed by using the zoom in or zoom out command. However, zooming is constrained by a set of predefined

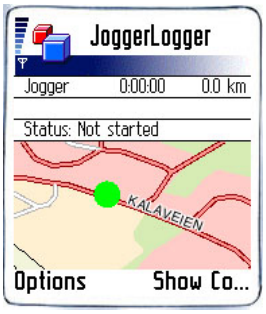


Figure 4.4: The browsing step for the task of creating a route.

zoom levels in order to prevent the map from getting too detailed or too general.

When the map is panned or zoomed, new areas will present themselves and the map data for these areas must be downloaded from the WMS. During the downloading, these new areas will remain black, but each area will be displayed as soon as the data is downloaded. The inner details of how the downloading is handled is explained in section 4.1.3, so please refer to this section for more information.

This step makes it possible for the user to become familiar with the surrounding if he or she is new to the area. Accordingly, this step is probably not that useful for users creating a route, but might be useful for users running routes that have been created by other users.

Logging Logging is the most import step, and it is at this point the tracking of the user takes place. This step is started by pressing the start command. At this point, the location of the jogger will be collected from the GPS via bluetooth once every five seconds.

When the logging step is entered, the user interaction with the map is changed from manual to automatic. This means that the user is no longer able to pan the map. Instead, the application will move the map according to the movement of the jogger. The reasoning behind the change in interaction is that the jogger will probably not be interacting with the map at all times during the jog. Instead, he or she will be more concentrated on the run, but occasionally looking at the application. Still, the map is zoomable.

In addition, two other changes are made to the GUI as well. First, the trail of the previous locations of the jogger is drawn on the map. Second, the time and the distance is showed in the white title line above the map. These changes can be seen in figure 4.5.

The user can stop the logging by pressing the stop command which will

Figure 4.5: The logging step for the task of creating a route.

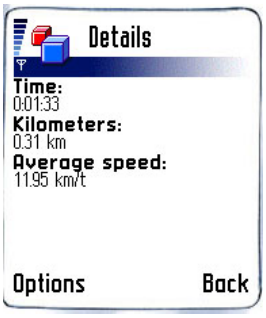


Figure 4.6: The showing details step for the task of creating a route.

take the user to the next step.

Showing details In this step, the user is presented with details about his or her run. The details about the run is kept relatively simple, and the user can only see the time used, the overall distance and the average speed, as showed in figure 4.6.

Saving This step involves, not surprisingly, the task of saving the information about the route and result that was created in the logging step. During saving, a progress bar, which can be seen in figure 4.7, is presented to the user which shows that saving is taking place. In addition, the user has the ability to cancel the saving by pressing the cancel command.

Run a route

The task of running a route is identical to the task of creating a route except that the route is already created. The information of the route is used by the application to enhance the user experience. Still, the enhancements are small, and the six steps explained in previous section remain more or less

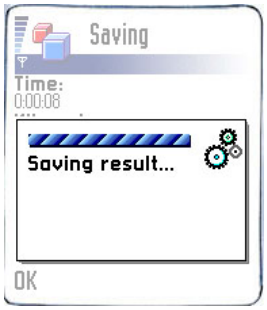


Figure 4.7: The saving step for the task of creating a route.

Figure 4.8: The initialization step for the task of running a route.

the same. Therefore, the explanation of the six steps will only include the enhancements that are made to them.

Initialization In this step, the user has to choose which route to run. This is performed by selecting one of the routes in the list of all the available routes, as figure 4.8 shows. This step differs completely from the same step for the task of creating a route where the user had to specify the name of the route.

Browsing This step is virtually the same as previously explained except that the outline of the route is showed as well. In this way, it is possible for the user to find the start of the route from the current location by using the map.

Logging This step is the same as the step explained for creating a route since it was not possible within the time limit for the project to include the proposed functionality of automatic stopping and off course notification.

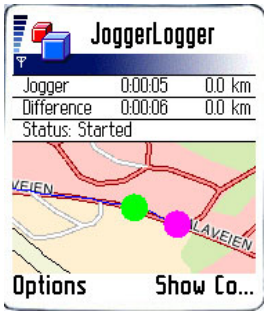


Figure 4.9: The logging step for the task of racing against an opponent.

Details This step is identical to the same step previously explained for the task of creating a route, so please refer to this task for more information.

Saving This step is identical to the same step presented for creating a route except only the result of the run is saved since the route already exist.

Racing against an opponent

The task of racing against an opponent is like the task of running a route except that the user jogs against someone. This means that the steps that both tasks consist of share a lot of similarities. In fact, the tasks are so similar that the steps browsing, details and saving are identical. Therefore, an explanation of these steps will not repeated in the following section when the different steps are presented.

Initialization The initialization that is needed before racing against an opponent is to choose the route to run and the opponent to run against. Both of these actions are performed by selecting the route and the opponent from a list of the available routes and results. This is performed in the same manner as a route is selected in initialization step for the task of running a route.

Logging In the logging step, the application presents and supplies the user with information about the opponent in three different ways. The first information is highly visual, and the opponent is positioned and rendered on the map in the same manner as the user is. This can be seen in figure 4.9 where the user is green and the opponent is purple. The position of the opponent is updated according to the time elapsed; Thus, the user is able to see whether the opponent is in front or lacking behind.

Figure 4.10: The mobile client synchronizing with the repository.

The second information is statistical, and it consists of the difference in time and distance between the user and the opponent. This information is displayed in the second line in white title line. The numbers in the first column are the difference in time and the numbers in the second column are the difference in distance. These numbers are negative if the user is lagging behind, and they are positive if the user is in front of the opponent.

Although visual information may be interesting, the user will probably not be looking at the screen frequently or at all. Accordingly, the application provides non-visual information as well by using vibration to notify the user about how the performance is compared to the opponent. A long vibration is given if the user is behind the opponent, and a short vibration is given if the user is in front. The notification is given at intervals according to the distance jogged where the current implementation uses an interval of every 100 meters.

Synchronization

Synchronization with the server is performed almost without interaction from the user. The only step needed is for the user to choose synchronization in the main screen. The user is then presented with the screen in figure 4.10 which tells the user which route or result that is currently being transferred. If the user by any chance wants to cancel, he or she can press the cancel command.

Currently there is a lack of flexibility since the user must send over all routes and results which have not been previously transferred to the repository. An improvement to be made here is to present a list of these routes and results to the user, giving he or she the ability to choose exactly which to send over. The selection can be made easy by providing convenience functions that select all and deselect all.

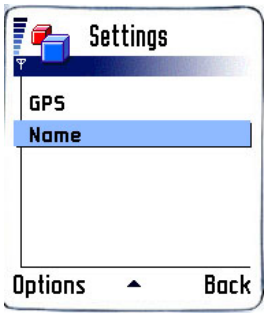
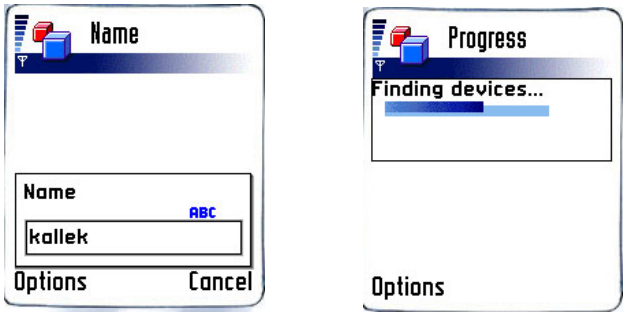


Figure 4.11: The list of settings that can be configured in the mobile client.



(a) (b)

Figure 4.12: The editors for textual settings (a) and bluetooth services settings (b)

Settings

The mobile client has a set of settings like most applications does. The settings are organized in a list, and a setting can be changed, as illustrated in figure 4.11, by first selecting the specific setting and pressing the change command. This will result in the specific editor for this setting being displayed. Currently, there are only two editors: a text area, which is used for settings containing textual information (see figure 4.12(a)); and a bluetooth service editor, which is used for selecting a bluetooth service (see figure 4.12(b)). The text area editor is used for the setting name, while the bluetooth service editor is used for setting the GPS service that should be used.

4.1.3 Details

The description of the underlying details of the implementation will only be given of the core parts of the mobile client. This is because a detailed description of all parts of the mobile client would be too extensive to include

in this paper as well as the different parts are not equally important or interesting.

Synchronization

The current implementation supports only one part of the synchronization. This means that it is only possible to send routes and results from the mobile client to the repository and not from the repository to the mobile client.

The sending of routes and results from the mobile client to the repository is performed using post requests over hypertext transfer protocol (HTTP). The mobile client will start by iterating over all the routes that have not already been transferred to the repository and it will send one route at a time. The information that is sent for each route is the route name, the time created, and the different coordinates that make up the route. The repository will try to store this information. If the repository is successful in storing the route or if the route already exists, the id of the route in the repository is returned. If, on the other hand, the repository fails to save the route for some reason, the text: "failed" is returned.

The client will naturally take different actions depending on the response received. If the response is the id of the route, this is saved in a file on the client in order to mark the route as saved. In addition, the id is needed when the results of the route are going to be transferred. However, if the response is failed, the client will skip the next step of transferring the results of the route.

Transferring a result follows the same steps as saving a route. First, the result is transferred to the repository, the repository either responds with the id of the result in the repository or the text failed. If the id is returned, this is stored on the client, while if it fails the mobile client tries to send the next result.

The marking of the routes and results are performed to minimize the traffic between the mobile client and the repository by only sending routes and results not already sent. However, marking is not necessary to prevent the repository from being cluttered up with duplicate routes and results since it is able to spot this.

Fetching location information

At first, the intent was to use the Location API for J2ME (JSR 179) to fetch location information since it supported all of the functionality needed for this project and more. However, the Location API is an optional part of J2ME and very few mobile phones on the market today support it. Still,

the Location API is not dead and it is getting more adoption. This can be seen in the recent update of the API to version 1.0.1 [12] and that Nokia, the biggest mobile company in the world, supports the Location API in their third edition the Nokia Series60 developer platform[13]. This is not of much help when the mobile phone used in development for this project only supports Nokia Series60 second edition. As a result, the Location API could not be used and the functionality of fetching the location information had to be implemented.

A decision was made to make the implementation a clone of the areas of the Location API which provided the functionality needed for this project. This decision was made for two reasons. First, the Location API specification could be followed and no time had to be used in designing the overall structure of the implementation. Second, it would be possible to swap the implementation with the Location API in the future when it has gotten wide adoption.

The main class in the implementation is the LocationProvider, and it basically includes all the code that fetch location information from the GPS. However, the class is a scaled down version of its counterpart in Location API and supports only three methods. These are getLastKnownLocation, which returns the lastKnownLocation; getLocation, which returns a Location within the time limit specified; setLocationListener, which registers a LocationListener that will be notified of the current location at the specified interval.

The LocationProvider class is implemented using two threads. The first thread is the reading thread which is responsible of communicating with the GPS through bluetooth. It will continually read the NMEA sentences sent from the GPS, extract the location information from them and update the last known location accordingly. If the connection to the GPS is lost for some reason, the thread will sleep for 5 seconds before it reconnects to the GPS. The second thread is the listener thread which will notify the registered LocationListener at a specified interval of the what the current location is. If the last known location is too old, it will mark the location as invalid.

Managing and visualization of maps

The managing and visualization of map information as well as managing and visualization of route, jogger and opponent tracks are handled by the RouteCanvas class which will be explained below. Some information about this class has already been given since this is the class that the user sees and interact with in the browsing and logging steps when creating a route, running a route and racing against an opponent. However, the inner details

and the design choices made in implementation has not yet been explained, and this will be the focus for this section.

Early on in the implementation of the RouteCanvas, there was a desire to use the Game API in MIDP 2.0 and its LayerManager class and Layer class to display map images, the outline of a route and the position and the track of a jogger and an opponent. At first, the intent was to create three subclasses of the abstract Layer class: one customized for the map, one customized for the route, and one customized for the results. However, this was impossible because it turned out that the constructor of the Layer class had package visibility, meaning only classes in the same package are allowed to extend it.

The next idea was to shoehorn the needed functionality into the two subclasses of the Layer class that already existed: Sprite and TiledLayer. The plan was to use a TiledLayer class for the map information and three Sprite classes for the route, jogger and opponent information. In order for this to be possible, the images that make up the Sprites had to be created on the mobile phone since the images change during a run. Furthermore, they had to be transparent, so that the different layers could be rendered on top of each other. Thus, the plan would be possible if it had been possible to create mutable transparent images in MIDP 2.0. However, in MIDP 2.0 it is apparently only possible to create either a mutable opaque image, or an immutable transparent image. As a result, the idea of using the LayerManager class and the Layer class was dismissed.

Still, the time invested in looking into the LayerManager and the Layer class was not a total waste since many of their principles are used in the actual implementation of the RouteCanvas. For instance, the notation of tiles is borrowed from the TiledLayer class, and the whole world map is sliced into tiles. However, this does not mean that all of this data is visible to the user or kept on the mobile phone at all time.

The area that is visible to the user is called the view port, a term borrowed from the LayerManager class, and this area is moved by the user when he or she pans. This will result in new tiles becoming visible, but the map data for these tiles might not be available for two reasons. Either the data has not been downloaded yet from a WMS, or it has been deleted from the cache used by RouteCanvas since the cache can only hold data for a specific number of tiles. In addition, the whole cache will be cleared if the zoom level is changed because this will change the level of details of the images.

The RouteCanvas uses a cache of four tiles where each tile has the same size as the view port. The reasoning behind these choices was to keep things simple and the set up has been successful in practice. However, in theory, the RouteCanvas should be able to support tiles and a cache of different sizes. The only limitation is that the current implementation does not support a

big sized cache well. This is because a new thread is spawned when a new tile needs to be downloaded. As a result, it is possible to have the same number of threads running as the size of the cache.

A better implementation would be to have a thread pool where the threads either wait for a task or performs one. For instance, the pool could be initialized to four threads that are allowed to download images from the WMS. In this way, the number of threads that are running at all time is limited, and a new thread does not have to be spawned for each task.

On top of the tiled map, the outline of the route, the jogger and the opponent are rendered. These parts are positioned according to the map, so that they are rendered in the right positions. The actual rendering of all the data happen in the following order: the map, the outline of the route, the jogger and the opponent.

4.1.4 Improvements

There are four improvements that is known about at this point, but there may be more. The first three are improvements needed to include all the functionality proposed in the design chapter. The last improvement, on the other hand, is needed to make the user experience of the mobile client better.

Automatic stopping of the mobile client must be included when the mobile client knows which route the jogger jogs. This can be implemented in two steps. The mobile client first checks if the jogger has jogged a distance approximately as long as the distance of the route. This is necessary since the stop location may be crossed more than once. Then the mobile client checks if the current location is in the area of the stop location, and if it is the logging is stopped.

The mobile client must notify the user if he or she is diverging from the route which is being jogged. This can be performed by checking whether a circle, with the location of the user as center and radius of for instance 25 meters, is intersecting with one of the line segments that form the route. If there is no intersection, the user is off course and he or she is notified by vibration. This calculation method will only work for small line segments, as in this application, since the curvature of earth can be disregarded.

The mobile client must also support downloading of route and result information from the repository. This can be performed by the mobile client sending a timestamp of when it most recently downloaded information from the repository. The repository will check which routes and results that have been added since and return ids for them along with a timestamp of when the request was made. These ids will then be used by the client in order to download only the newly added routes and results.

Error handling must be better on the mobile client and this apply especially to errors connected to fetching location information from the GPS receiver. In the current implementation, the user will receive no notification if the GPS is either unreachable or if the GPS is not able to determine the current location.

4.2 Repository

The implementation of the repository has support for storing and fetching basic information about routes, results and joggers. This information actually includes more than what is needed by the mobile client and web client in the current implementation. Still, there are areas of the repository that lack much of the functionality described in the design chapter. These areas are mostly connected to the limited functionality of the web client which the repository is responsible for serving with web pages and performing actions in response to input from the user.

A more detailed description of the implementation of the repository will be given in the next sections. First, an explanation is presented of why the technology used in implementing the repository was chosen. Then a short description is given of how the user will come in contact with the repository. Next, the overall architecture of the whole implementation is presented along with how the different parts of it is implemented. In the end, the improvements needed to be made to the implementation of the repository is outlined.

4.2.1 Technological considerations

There were basically two different technologies that were possible to use in the implementation of the repository considering my knowledge of programming languages and web development. These two were PHP: hypertext preprocessor (PHP) and Java. PHP is the one of the two that I have used the most for web applications previously. It is easy to use and it is fast to develop and create web pages. However, the PHP code can easily become unstructured if the developer is not careful. In this area, Java is probably a better choice. In addition, Java has a wide range of frameworks that make the structuring and separation of the different parts even easier. However, there may be similar frameworks for PHP, but I am not accustomed to any of them. As a result, Java was chosen for its structural strengths and its good framework support.

The implementation of the repository was tested on the servlet engine Tomcat 5.5 and the Rational database management system (RDBMS) Post-

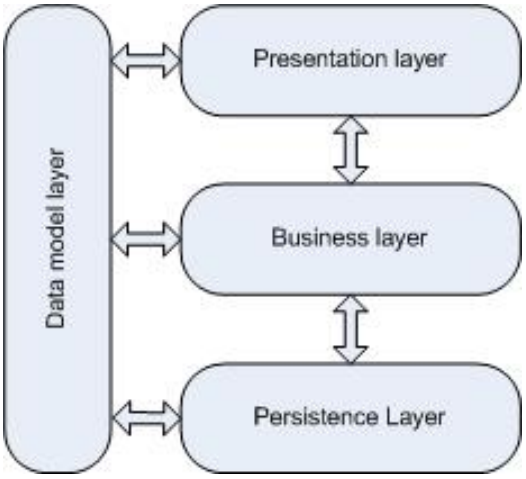


Figure 4.13: The different layers that form the repository and how they are connected each other.

greSQL 8.1 . However, it should run in any servlet container which support servlet version 2.4 and Java Server Pages (JSP) version 2.0. In addition, it should be possible to change the RDBMS by making some small changes to a set of configuration files as long as the RDBMS has support for transactions.

4.2.2 User interaction

The user only interacts with the repository indirectly using either the mobile client or the web client. The mobile client communicates with the repository when performing synchronization. The web client, on the other hand, is more tightly coupled to the repository, since it is the repository that actually delivers the web pages which make up the web client.

4.2.3 Details

The repository consists of four different layers as illustrated in figure 4.13. These four layers are the domain model layer, the persistence layer, the business layer and the presentation layer. The domain model layer contains the classes that are needed to hold the information about the routes, results and joggers in the repository.

This layer is used by the other three layers to exchange data between each other and perform actions on this data. The persistence layer is responsible for persisting and fetching information from a persistence storage. However, this layer does not include business and transactional logic, this is instead added by the business layer. The presentation layer uses the business layer

to fetch the information that should be presented to the user or to store the information entered by the user.

Each layer communicates with each other using interfaces instead of classes. As a result, the underlying implementation of each layer can be swapped without affecting the other layers. Currently, there is only one implementation of each layer and these implementations will be discussed in the next sections.

Domain model layer

The domain model layer contains five classes that are needed to represent the information in the repository. These five classes are Jogger, Route, Result, Coordinate and Location. How each of these classes relate to each other is illustrated in figure 4.14.

The Jogger class is used to represent the users of JoggerLogger, and each instance will contain the following demographic information: first name, last name, username and password. The demographic data could have been expanded, but none of the other layers need this information in this light weight implementation. Furthermore, each jogger will be connected to a set of routes and results through the properties routes, results, favorite routes and opponents. The first two are the routes and results which the jogger has added to the repository. While the other two are the routes and results the jogger finds interesting and wants to transfer to a mobile phone using the mobile client.

The route class is used to represent the routes that have been created, and each instance will have a name, a creation time as well as a list of coordinates that form the outline of the route. The list is ordered, so that each coordinate has an index of which item it is in the list along with positional information in the form of longitude, latitude and altitude. In addition, each route will have a set of results which have been achieved by the users jogging the route.

An instance of the result class consists of information about one of the results in the repository. It will contain when the result was created and an ordered list of the locations the jogger visited during the creation of the result. Since the list is ordered, the Location class, like the coordinate class, includes an index property. In addition, the Location class includes information about the position and the time the position was collected.

Persistence layer

The persistence layer has three interfaces, RouteDao, ResultDao and JoggerDao, which specifies how the domain model objects can be persisted and

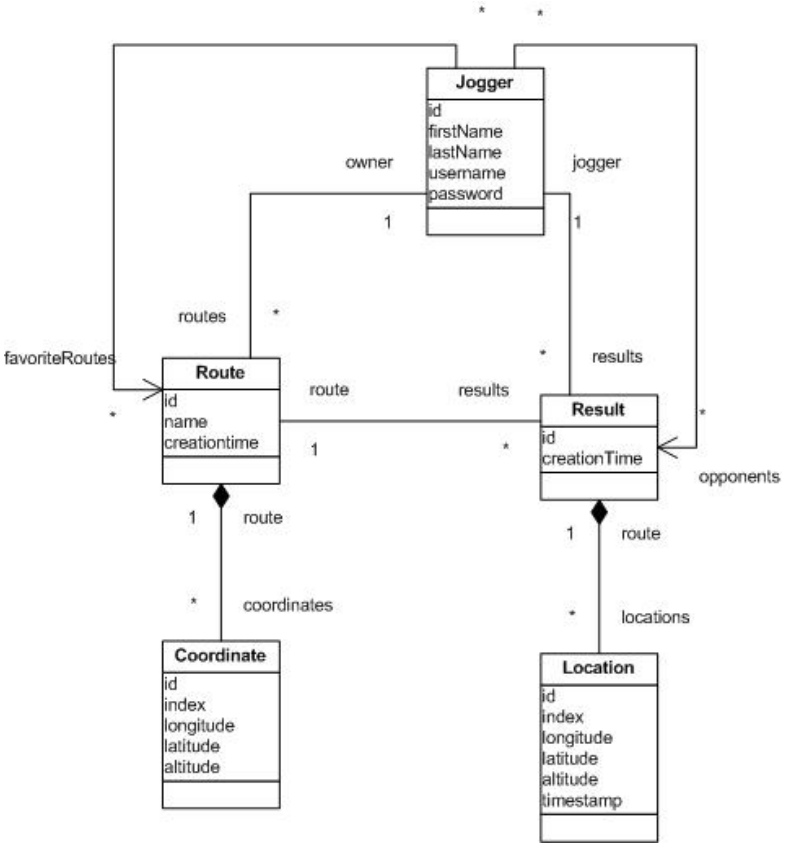


Figure 4.14: UML diagram of the domain model layer in the repository.

fetched from a persistent storage. The actual implementation of these interfaces and the layer as a whole are done using a object/relational mapping (ORM) tool called Hibernate [14].

Hibernate, as all ORM solutions, allows an object model to be persisted and fetched from a RDBMS by mapping classes to relations [15]. However, the problem with this mapping is that relations and classes do not map one to one, meaning that some conversion must be done between the two representations. The great thing about Hibernate and other ORM solutions are that they can handle this conversion so that the developer can concentrate on other tasks.

Hibernate also supports various RDBMS and makes it easy to change which one that is used by making small changes to a configuration file. In the current implementation, the RDBMS PostgreSql is used since it is open source and supports the needed functionality of transactions.

Business layer

The business layer consists of three interfaces, RouteService, ResultService and JoggerService, which specify the same methods as the interfaces in the persistence layer. This is because the same actions should still be possible only that business and transaction logic are added to them. In the current implementation of this layer, each service class composes its DAO counterpart and adds transactional support to the methods. As a result, it is possible to make several method calls across different service objects atomic, meaning either all methods are performed or none at all. This is performed by using the Spring framework [16] in two ways.

First, Spring is used to populate the different fields of the service objects with instances of the DAO counterparts. This action is called wiring, which is a part of the core functionality of Spring, and allows developers to use dependency injection. Dependency injection basically means that instead of an object populating a field itself, some other object performs this task [17]. In this way, it is easy to swap one implementation of one of DAO interfaces with another one.

Second, aspect oriented programming (AOP) in Spring is used to add transactional support to the service classes. "... AOP decomposes programs into aspects or concerns", which "... enables modularization of concerns such as transaction management that would otherwise cut across multiple objects." [18]. As a result, functionality can be added to classes without making any changes to the actual source code of these classes. In Spring, AOP is configured using a XML file.

Presentation tier

The presentation layer uses the Spring MVC framework, which is a part of the Spring framework used in the business layer. This framework enables developers to create web applications by taking advantage of Spring's wiring functionality and the Model-View-Controller (MVC) design pattern.

MVC is a popular design pattern for graphical user interfaces and splits the presentation into three parts: the Model, the Controller, and the View. The model represents the information in the application, the view renders this information by using the model and the controller changes this information by performing actions on the model according to the input received from the user [19]. The advantage of using this pattern is that the model can be easily reused and that it is easy to add more views.

The implementation of this design pattern for the repository will be explained in the following paragraphs.

Model The model consists of seven classes all in all. The first five classes are the classes in the domain model layer. It is possible to use them since the only requirement for the model objects in Spring MVC are that they conform to the Java beans standard of defining properties using get and set methods. The last two classes are the RouteCommand and ResultCommand which are used to convert coordinates and locations entered as text to actual Coordinate and Location objects.

Controller The controller is limited in functionality and consists of only four controllers. These are the AddRouteController, which allows a user to add a route; the AddResultController, which allows a user to add a result; the ListRoutesController, which lists all the routes; and the GetRouteController, which displays a Route. More controllers needs to be implemented for a full implementation, but these four controllers are enough to support synchronization using the mobile client and validate that this synchronization actually succeeded.

View The view actually consists of two different kinds of views which both use JSP as view technology. The first is used for the mobile client when synchronization is performed. In this layer, "failed" will be displayed if the AddRouteController or AddResultController do not succeed in adding a route or result and id of the route or result is displayed otherwise. The view for the other two controllers is empty, since they should not be used by the mobile client. The second is used to create web pages for the different controllers,

and these web pages make up the web client. How these web pages look and how the user can interact with them will be discussed in section 4.3.1.

4.2.4 Improvements

The improvements needed to the repository are mostly connected to the presentation layer. The presentation layer must be expanded to include more controllers and views that incorporate the functionality of the web client. However, it may be necessary of changing some of the other layers if presentation layer needs some functionality that are not available yet.

4.3 Web client

The web client is the part of JoggerLogger that has received the least attention since it is the least needed part. This can be seen in the number of features it includes. Still, the web client has served its purpose of showing that it is possible for the mobile client to transfer information to the repository and then later view this information in the web client.

Much of the information in the web client is shown on a map which is fetched from the map service: Google Maps [20]. This service includes more functionality out of the box than WMS. However, the fine grained control available with WMS is not present in Google Maps, but this is not needed in the web client at this point or may not be needed at at all.

In the next section, the GUI of the web client is presented before the improvements needed to the web client are discussed.

4.3.1 User interaction

The user can perform four actions in the current implementation of the web client. These are adding a route, adding a result, list routes and view a route, and a description of all of these actions will be presented in the next sections.

Add route

The user is able to add a route through the web page presented in figure 4.15. This web page has three input fields for the name of the route, the creation time, and the coordinates that form the route. If the format of some of the fields are wrong an error message is shown to the right of the field.

This page is mostly created for debugging purposes to check if it possible to store a route in the repository. In a full implementation of the web client,

Figure 4.15: The web page for adding a route.

this page would probably use a file field where the user could upload a route file instead of entering it manually.

Add result

The task of adding a result is similar to adding a route, and much of the information presented in the previous section apply to this action as well. The only exception is that the web page include some other fields which are presented in figure 4.16. The user must enter the creation time of the result, the route it belongs to and the locations of its track.

List routes

The user is presented with a map where the start location of all the routes in the repository are marked. In addition, all the routes are listed in the right column. This is illustrated in figure 4.17. The user can view a route either by clicking on one of routes in the right column or on the link presented when clicking on one of markings in the map.

View a route

The user is presented with a map over the area of the route. The outline of the route is drawn on the map, and the top 10 results are listed in the right

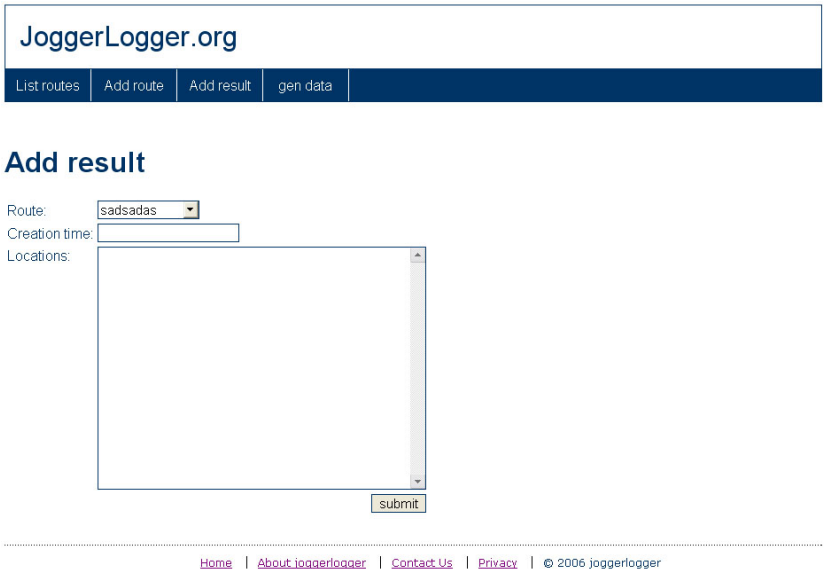


Figure 4.16: The web page for adding a result.

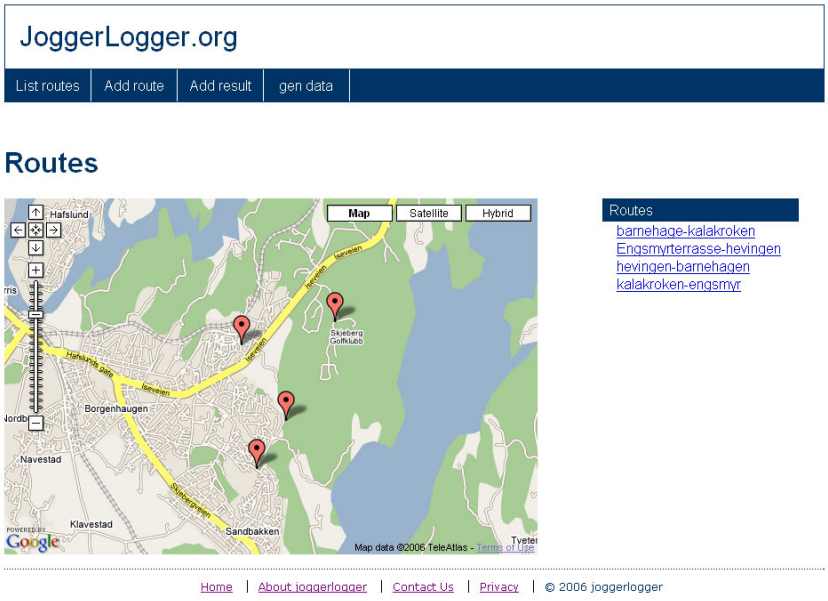


Figure 4.17: The web page which lists all the routes in the repository.

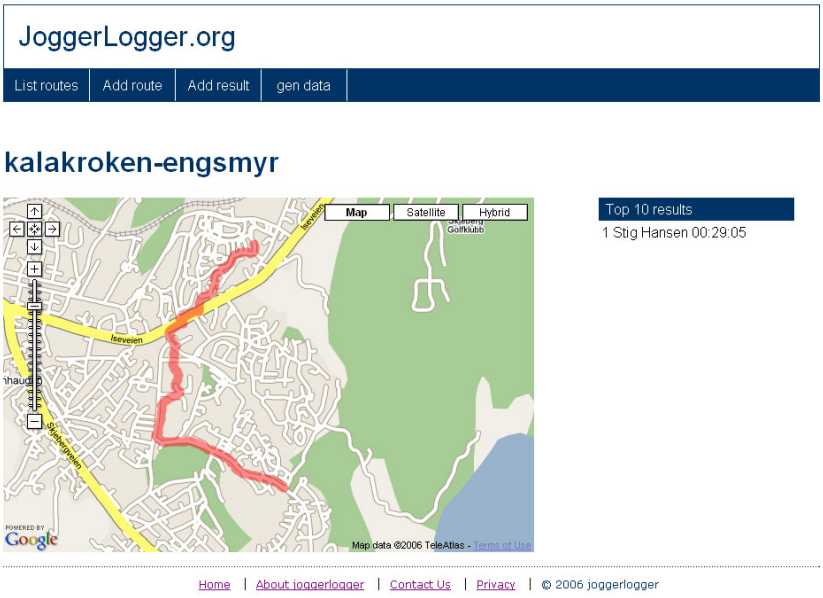


Figure 4.18: The web page which presents one of the routes in the repository.

column, as shown in figure 4.18.

Improvements

The web client needs a lot of improvements in order to add support for the functionality presented in the design chapter. It supports almost all the functionality of finding routes. The only part that is missing is for the user to be able to add the routes to their favorites in their account. On the other hand, the other two actions of managing account information and replaying results are not supported at all and must be added.

4.4 Summary

JoggerLogger has been realized partly in a proof of concept implementation since a full implementation was not possible in the available time. The main part of the system, the mobile client, has received the most attention, and it supports almost all of the planned functionality. This includes creating a route, running a route, and racing against an opponent as well as one-way synchronization.

On the other hand, the implementation of the other parts lack much of the proposed functionality and were developed mainly to show that it is possible to transfer routes and results from the mobile client to the repository. Storing

and retrieving jogging information is supported by the repository, but much of the functionality needed by the web client is not included. The web client has received the least attention of the three parts, and it is only possible to see which routes and results that are stored and add routes and results.

In conclusion, the proof of concept implementation has served its purpose and showed that it is technological feasible to implement JoggerLogger.

Chapter 5

Testing

Usability testing was performed on the mobile client of JoggerLogger with the goal of finding areas that are problematic for the user and how they might be improved. The focus was to determine if it would be possible for a jogger to use the mobile client when jogging without help from others. It was tested on three joggers of varying in age, sex, mobile and computer experience, and led to a set of discoveries and improvements which will be detailed later in this chapter.

All parts of the JoggerLogger system should have been tested in order to see if they work in a satisfactory manner. However, this is not possible since the whole JoggerLogger system is not fully implemented, and there is limited time available. Therefore, a decision was made to test only one part of the system, the mobile client.

Choosing to test the mobile client instead of the other parts of the system was easy to make for three reasons. First, it is the part of the system that is most important, so if this part is not working properly this will make the other parts almost useless. Second, this part has almost been fully implemented where the implementation of the other parts still lack much more of the planned functionally. Third, the limitations of mobile phones in form of limited screen size and cumbersome input methods, makes the need for testing even more important than for applications running on a computer.

The user group for the mobile client as well as the JoggerLogger system as a whole is joggers of all ages. The only requirement for the user is that he or she has basic skills in using the platform where the different clients run. For the mobile client this means that the user must be able to navigate in the mobile phone and write text.

5.1 Test objectives

The main objective of the test is to determine whether the three main tasks in the mobile client, creating a route, running a route, racing against an opponent, are easy to use. However, this objective is too general, making it difficult to determine exactly which areas that are easy to use and which areas that need improvements. Thus, the main objective has been split into the following sub objectives:

- Is the user able to zoom and pan the map presented?
- Is the user able to start the logging of the mobile client?
- Is the user able to understand the information presented in the title line in the upper area of the mobile client?
- Is the user able to understand the vibration scheme while racing?
- Is the user able to stop the logging of the mobile client?

5.2 Test group

The test group consists of three joggers who vary in sex, age, computer experience and mobile experience. Thus, the test group represents the user group of the mobile client and JoggerLogger in a sufficient manner, although it would be preferable with a bigger test group if more time had been available. The actual characteristics of the three participants can be seen in table 5.1 below.

5.3 Test procedure and tasks

Before the test was performed, each person was given some background information about the test and the mobile client. First, the test person was informed about the objectives of the test, how the test would be conducted and how the results would be collected and presented. The goal was to put the test person at ease by explaining that the goal was not to test them, but the mobile client. Second, the test person was given a short introduction to the mobile client and the goals that it tries to achieve.

During the test, each test person was studied by an observer who took note of different discoveries. However, the observer did not interact with the test person in any way unless the test person had problems that would

Table 5.1: Characteristics of the test group

Characteristic	Choices	Test persons
Sex	Male	2
	Female	1
Age	<20	
	20-40	2
	41-60	1
	>60	
Computer Experience	inexperienced	1
	normal	2
	experience	
Mobile phone experience	inexperienced	1
	normal	2
	experience	

make it impossible to continue the test without help. In this way, the test environment would closely mirror the reality where the user is left more or less alone, and the results could be compared more closely since the test environment was kept as neutral as possible.

After the test, the observer had an informal talk with the test person. This talk was used to get more information about the discoveries made during the test, and to receive feedback about which parts of the mobile client that could be improved.

The actual test consisted of three tasks that the test person had to perform. The tasks were bound to the three main actions in the system which are creating a route, running a route, and racing against an opponent. How these tasks were structured will be presented in the next sections.

5.3.1 Create a route

In this task, the test person was asked to create a route by following the five steps explained in the implementation chapter. The test person was to start by choosing to create a route in the main screen, and then choose an appropriate name for the route. Next, the test person would start jogging and inform the application of this. During the jogging, the test person was asked to look at the screen occasionally, so that it would be possible to ask questions about the information that was presented. When the test person

reached the end of the route, he or she was to inform the mobile client of this. Lastly, the test person would save the route and result that had been created.

5.3.2 Run a route

In this task, the test person was to run a route that had already been created. Before starting to run, the test person was to pan the map to see the whole outline of the route that was presented on the map. In addition, he or she was to zoom in and zoom out of the map. Then, the test person would jog the route, informing the mobile client of when he or she started and reached the end. This task was ended by the test person saving the result.

5.3.3 Race against an opponent

In the third task, the test person would run against himself or herself. This meant the test person would select the route and one of the results which had been recorded in the two previous tasks. Again, the test person was to start jogging and inform the application of this. However, this time the test person was asked to look closely at the screen, and see if he or she was able to determine what the different pieces of information were. The next steps that had to be conducted by the test person, which were the same as the steps for the other two tasks, were informing the application that the run had ended and saving the result.

5.4 Test results

All of the test persons managed to create a route. There were no problems in specifying the name of the route, start and stop the logging and save the route and result. However, two of the test persons were unsure if the mobile client had started logging or if they had to. This problem was resolved by both when they noticed the text "not started" in the title line.

The second task was successfully completed by all test persons. They seemed more confident and hesitated less when performing the different actions. This might be a result of the test persons remembering what they did in the previous task. However, one of the test persons had problems with stopping the logging. Not because the test person did not remember what to do, but because he or she believed the mobile client used the information it had about the route to stop the logging automatically when reaching the end.

In addition, the test persons had problems with panning and zooming the map that was presented. One of the test persons tried to pan the map after the mobile client had started logging, and did not understand that the mobile client handled the panning instead of the user itself. The problem with zooming in and out of the map was based on the fact that the test persons believed the button in the middle of the directional buttons is used for this tasks. Still, all of the test persons were able to figure out that the zoom in and zoom out commands should be used instead.

The third task was completed by all test persons just like the other tasks. All the test persons managed to interact and understand the information displayed about the difference in time and distance. Nevertheless, two of the test persons had problems determining which of two circles displayed on the map that were the jogger and the opponent.

5.5 Improvements

The results from the test were promising and most of the mobile client was easy to use. However, there were some areas that were discovered that could be improved. These improvements are presented and discussed in the next paragraphs.

1. The positioning of the map in the browsing and logging step should probably be changed, so that user, and not the mobile client, controls whether the positioning is handled by the user or the mobile client. This can be done by adding commands that toggle the interaction mode and presenting information that shows which interaction mode that is currently used. As a result, the user has better control over the mobile client and does not have to be confused of the mobile client changing the interaction mode automatically.

2. The representation of the jogger and opponent should be changed in order to present which direction they are headed and to better show which of the two is the jogger and which is the opponent. Direction can be easily visualized by representing the jogger and the opponent as triangles which point in the direction they are moving. On the other hand, it is more difficult to propose a change in the representation that will make it easier to see which is whom. One possibility is to number them, so that the jogger is always number one and the opponent is number two. However, this can be misunderstood as the position they have compared to one another. Another solution, is to write in the title line that purple is the opponent and green is the jogger.

3. The mobile client should be able to stop automatically when running a route and racing against an opponent. This functionality has already been explained in the design chapter, but there was no time to include it in the proof of concept implementation.

4. The mobile client should probably include a better visualization of whether logging has been started or not. This can be done by removing all the information that is not important in browsing like time and distance. As a result, the text: "status: Not started" could be made bigger and easier for the user to see.

To sum up, the usability test showed that the three main tasks, creating a route, racing a route and running against an opponent, works in satisfactory manner, and the tests persons were able to perform all of them. However, the test persons had several problems with each task and there are room for improvements. These improvements include automatic stopping of the logging, more control of the positioning of the map and better visualization of the jogger, opponent and whether the logging has been started.

Chapter 6

Conclusion

The goal of this project has been to use GPS and standard web technology in order to enhance the jogging experience. This has been proved to be possible from a technological view point through the proof-of-concept implementation of JoggerLogger. It basically showed that it is possible to create routes and results on a mobile phone by fetching location information from a GPS receiver, and share these routes and results by transferring them over the Internet to a publicly, available repository.

In addition, the proof-of-concept implementation and the usability test performed on it showed that it is possible to implement the project so that it is understandable for the user. This was achieved by focusing on the user through out the whole development and being aware of the limitations of the different platforms used. Especially the limitations of mobile phones was taken into consideration in order to keep the GUI as clean and understandable as possible.

Still, there is one concern of JoggerLogger that has not been addressed at all and this is if it will be attractive for people to use. Answering this with a definitive yes or no is impossible without performing extensive market research. Instead, I will present the three reasons why I think this system has great potential.

First, there are a lot of people that jog and train, probably more now than ever before considering the increased focus training and dieting has gotten in media in recent years. As a result, there are a lot of people than can be attracted to such a system.

Second, there are many commercial solutions offering similar functionality to joggers already. Thus, it is reasonable to believe that this something joggers actually want. This is further strengthen by Nokia's recent release of the Nokia 5500 Sport mobile phone which is targeted at joggers [21].

Third and lastly, mobile phone technology evolve rapidly and mobile

phones include more and more technology found in other devices. GPS is an example of such a technology which will be included, and some mobile phones include it already. With the inclusion of GPS in the mobile phones, the JoggerLogger system would be more attractive since a separate device does not have to be purchase and carried around during running. In this way, the threshold of trying the mobile client is less and the user experience will be better.

To sum up, it is technologically possible to implement JoggerLogger and to do so in a way that is understandable. In addition, it is reasonable to believe that joggers will be attracted to it. However, the description of the project has only scratched the surface of what is possible with today's technology and this will only increase in the future with the rapid growth of mobile, web and location technology.

Bibliography

[1] "frwd technologies - sports technology - training tool - outdoor computer - heart rate monitors - running computer - etusivu," FRWD Technologies Ltd, Mar 2006. [Online]. Available: http://www.frwd.fi

[2] "Tractrac," Trac Trac, Mar 2006. [Online]. Available: http://www.tractrac.com/

[3] "Sportsim - share and experience," Sportsim, Mar 2006. [Online]. Available: http://www.sportsim.com

[4] "Mobile phone - wikipedia, the free encyclopedia," Federal Aviation Administration, May 2006. [Online]. Available: http://en.wikipedia.org/wiki/Mobile_phone

[5] P. H. Dana, "Global positioning system overview," The University of Colorado at Boulder, May 2006. [Online]. Available: http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html

[6] "Nationwide differential global positioning system program fact sheet," Federal Aviation Administration, May 2006. [Online]. Available: http://www.tfhrc.gov/its/ndgps/02072.htm

[7] *OpenGIS Web Map Server Implementation Specification*, Open Geospatial Consortium, Inc., May 2006. [Online]. Available: http://portal.opengeospatial.org/files/index.php?artifact_id=14416&passcode=x3vuw4b3hu7p1n8y08ae

[8] "Google maps," Google, May 2006. [Online]. Available: http://maps.google.com/

[9] "del.icio.us," Yahoo, May 2006. [Online]. Available: http://del.icio.us/

[10] "Welcome to flickr - photo sharing," Yahoo, May 2006. [Online]. Available: http://flickr.com/

[11] A. E. Hansen, "Mobuddy - making mobile messaging easy and fun," Østfold University College, May 2006. [Online]. Available: http://freja.hiof.no/digmap/resources/mbuddy_poster.pdf

[12] "Jsr 179 location api for j2meversion 1.0.1," Nokia Corporation, May 2006. [Online]. Available: http://www.forum.nokia.com/info/sw.nokia.com/id/f1957ea1-5a79-406e-be49-306cfd15d2da.html

[13] "S60 2nd/3rd edition: Differences in features v1.3," Nokia Corporation, May 2006. [Online]. Available: http://www.forum.nokia.com/info/sw.nokia.com/id/36e49fcd-b9d5-4c5f-985c-f4ec5ae76dc2/S60_2nd_3rd_Ed_Differences_in_Features_v1_3_en.pdf.html

[14] "hibernate.org - hibernate," JBoss Inc., May 2006. [Online]. Available: http://www.hibernate.org/

[15] *HIBERNATE - Relational Persistence for Idiomatic Java*, JBoss Inc., May 2006. [Online]. Available: http://www.hibernate.org/hib_docs/v3/reference/en/html_single/

[16] "springframework.org," springframework.org, May 2006. [Online]. Available: http://www.springframework.org/

[17] M. Fowler, "Inversion of control containers and the dependency injection pattern," May 2006. [Online]. Available: http://www.martinfowler.com/articles/injection.html

[18] R. Johnson, J. Hoeller, A. Arendsen, C. Sampaleanu, R. Harrop, T. Risberg, D. Davison, D. Kopylenko, M. Pollack, T. Templier, and E. Vervaet, *Spring - Java/J2EE Application Framework*, springframework.org, May 2006. [Online]. Available: http://www.springframework.org/docs/reference/

[19] "Java blueprints - j2ee patterns," Sun Microsystems, Inc, May 2006. [Online]. Available: http://java.sun.com/blueprints/patterns/MVC-detailed.html

[20] "Google maps," Google, May 2006. [Online]. Available: http://maps.google.com/

[21] "Nokia 5500 sport: Smartphone with a six-pack," Nokia Corporation, May 2006. [Online]. Available: http://press.nokia.com:80/PR/200605/1050231_5.html

List of Figures

2.1 An example of a request to a WMS using the GetMap operation. 6

3.1 Overview of the three parts which JoggerLogger consists of. . 9

4.1 the main screen of the mobile client. 19
4.2 The five steps that are needed to create a route. 20
4.3 The initialization step for the task of creating a route. 20
4.4 The browsing step for the task of creating a route. 21
4.5 The logging step for the task of creating a route. 22
4.6 The showing details step for the task of creating a route. . . . 22
4.7 The saving step for the task of creating a route. 23
4.8 The initialization step for the task of running a route. 23
4.9 The logging step for the task of racing against an opponent. . 24
4.10 The mobile client synchronizing with the repository. 25
4.11 The list of settings that can be configured in the mobile client. 26
4.12 The editors for textual settings (a) and bluetooth services settings (b) 26
4.13 The different layers that form the repository and how they are connected each other. 32
4.14 UML diagram of the domain model layer in the repository. . . 34
4.15 The web page for adding a route. 38
4.16 The web page for adding a result. 39
4.17 The web page which lists all the routes in the repository. . . . 39
4.18 The web page which presents one of the routes in the repository. 40

List of Tables

5.1 Characteristics of the test group 44