

FREE UNIVERSITY OF BOLZANO



Route Tagging

Internet and mobile services

Academic year 2009/10

Michael Dejori

Stefan Peer

1. MOTIVATION.....	3
2. SYSTEM FUNCTIONALITIES	4
2.1 CLIENT	4
2.1.1 Route tagging	4
2.1.2 Saved routes – My routes	4
2.1.3 Search routes.....	4
2.2 SERVER.....	5
2.2.1 Storing routes.....	5
2.2.2 Uploading routes.....	5
2.2.3 Downloading routes	5
2.2.4 Web interface.....	5
3. SYSTEM ARCHITECTURE.....	6
3.1 CLIENT ARCHITECTURE	6
3.1.1 Recording Route.....	6
3.1.2 Route Object.....	7
3.1.3 Store Routes.....	7
3.1.4 Connection Manager.....	7
3.1.5 Route GUI.....	7
3.2 SERVER ARCHITECTURE.....	8
3.2.1 Beans.....	8
3.2.2 DB	8
3.2.3 Routeservice	8
4. HUMAN/COMPUTER INTERACTION.....	10
4.1 CLIENT-SIDE INTERACTION	10
4.2.1 Home screen.....	10
4.2.2 Recording a new route.....	10
4.2.3 My Routes.....	12
4.2.4 Search by keyword.....	14
4.2.4 Search near location	15
4.2.5 Downloaded Routes.....	15
4.2 SERVER-SIDE INTERACTION	15
5. GOOGLE MAPS API.....	17
5.1 KML	17
5.2 STATIC IMAGE MAP.....	18
5.3 INCLUDE MAP AND DESCRIPTION IN WEB SERVICE	18
6. MAJOR TECHNICAL PROBLEMS AND RELATED SOLUTIONS.....	19
7. CONCLUSION.....	19

1. Motivation

Route Tagging is an application that should allow users, which are interested in rambling and walking, to share their routes with other users. For such a “sharing application” it is not only necessary to write the application, but another key issue for making the application good is that there is the need of many users which share their data, so that other users can make use of it.

Developing an application for a handheld device is partly a new experience for us. Mobile phones allow us to include also technologies like GSM which we didn't use so far. We decided to develop *route tagging* because we wanted to involve the location API and it is an interesting application which could be used also in real life. Imaging, you are in a foreign village and want to ramble with your family. You could take your phone, search for a route which is near your actual position and download it together with some information. Together with the information like the distance, altitude difference and so on you get also a description, grate and rating of another user which can be useful for you.

2. System functionalities

The whole “Route Tagging” system is divided into two parts: a server and a client part. The server is based on JSP and runs as a web service. The client is based on J2ME and is able to connect to the server, using the whole functionality of the system, or can also stand alone (without a server).

2.1 Client

2.1.1 Route tagging

This is the main functionality of the system and allows a user to store information about a route he walked. The tagging of a route begins with a first snapshot of the users’ current position and time 0. Then in fixed intervals are taken position and time snapshots which are associated to the route. At the end of the tagging a last position/time snapshot is made. Finally a route consists of an initial and an end position and of several positions the user passed during his walk (each position at a given time). So the route can be easily reconstructed afterwards. By this geographical information and the time, there can be calculated several statistics of the route which are given by a summary of:

- Time spend
- Length
- Altitude difference
- Average velocity

Using “Google Maps” the route is also shown in a small map, so that it’s clearer for the user.

In addition to this information, the user can also give a title, a description, a grade and a ranking to the route, so that he or others know afterwards something about it. The grade explains how difficult the route was and the ranking determines how the user liked it.

After the tagging of the route, it is stored locally on the phone and the user has also the possibility to upload it to a server, so that it will be shared (more on that later).

2.1.2 Saved routes – My routes

All routes of a user are stored locally on the phone, such that he can retrieve their summaries. He is also able to delete them.

2.1.3 Search routes

The system provides also a “Route tagging web service”, which allows users to share their routes among others. There can be searched among all uploaded routes. Search constraints are either the users’ actual position or the routes’ title or description.

If a user found a route he liked, he can download it, so that it will be stored locally on his phone and he can always access it.

2.2 Server

The main functionality of the server is to manage routes of all users and provide an interface for the client so that users can upload and download routes.

2.2.1 Storing routes

The server stores all routes and all waypoints of routes in a database.

2.2.2 Uploading routes

The server provides an interface (Servlet) to the client so that a route, with all its waypoints, can be uploaded. The servlet stores the information it gets in the database.

2.2.3 Downloading routes

Routes can be downloaded either by specifying a location or a keyword. The web service offers for both options a different servlet, which retrieves the required information from the database and returns it to the client.

2.2.4 Web interface

The “Route tagging web service” offers also a web interface (JSP/HTML), which allows users to access/search routes using a browser. For each route there is also shown a map and its summary.

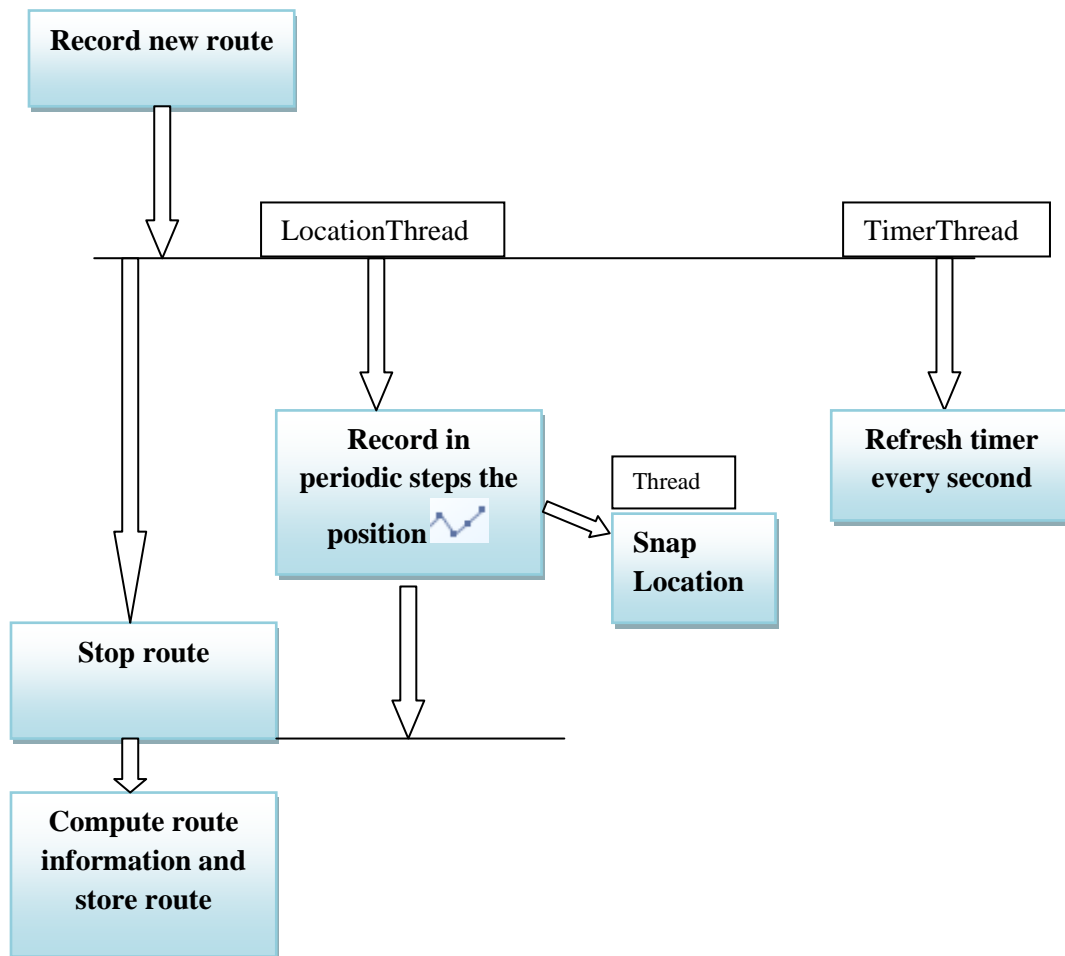
3. System architecture

3.1 Client architecture

3.1.1 Recording Route

When developing an application in Java 2ME threads are especially important, because there are many operations which require a few seconds and we don't want that the user interface is freezing for that time. Moreover, the application has an integrated timer which would stuck the application or not be synchronized.

The following figure will help us understanding the architecture of the functionality "recording a route".



When the user wants to tag a new route, then there will be created two threads:

- **LocationThread:**
It is responsible for asking in periodic steps (intervals) the actual position of the user. When a location must be retrieved, a new thread "SnapLocationThread" is instantiated, which uses the location API for asking the location. Again this new thread is needed because asking the location needs a few seconds. Otherwise the interval would be increased of the time needed for asking the location API.
- **TimerThread:**
It is responsible for refreshing the timer on the screen, which shows to the user how long he is walking.

3.1.2 Route Object

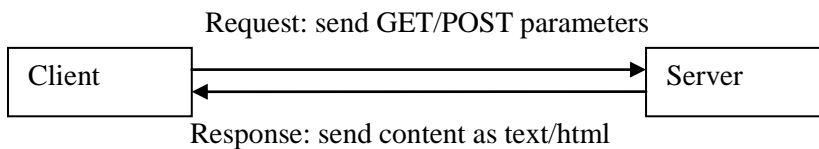
We created a class RouteObj which represents all information of a route and a list of its location snaps. This class offers also a method to convert the object in a byte array, so that it can be stored in a record store. There exists another method for the way back, out from the store. The RouteObj has also some methods, which are able to calculate some route statistics according to waypoints (location snaps) such as altitude difference, distance and average speed.

3.1.3 Store Routes

All tagged routes are stored in a record store, so that they are available persistently on the phone and do not disappear when the application will be closed. We created our own class "RouteRecordStore" which can easily add, delete, modify and retrieve routes from the record store. Since we have 2 different record stores for downloaded and tagged routes (my routes) we easily can maintain 2 instances of this class.

3.1.4 Connection Manager

We created the connection manger class for all communication which happens between client and server. This class offers methods, which allow the client to request information (text and images) from the server, such as searching and downloading routes but also to upload a route. Since on the server side there is a HTTP web service, there may be used GET or POST parameters for sending data from client to server. Our class is able to handle both types. Communication from server to client works in sending content as text/html (like a webpage).



A very important issue in this class is the URL encoding. Since parameters may contain special characters, they have to be encoded such that communication works. We provided our own class "URLEncoder" which does this for us.

3.1.5 Route GUI

The class "Route" is the central class in this application and connects quite all other classes. It manages the whole graphical user interface of the J2ME application and maintains instances of record stores, route-record threads, the connection manager, etc. There are also all the forms, commands, listeners, alerts and other GUI elements.

3.2 Server architecture

The main goal of the server is to store routes and waypoints in a database and providing an interface to clients such that they can retrieve information from the database. We used the MVC model for handling data: servlets extract data from the database and store it into beans. The beans make it available for the website (JSPs).

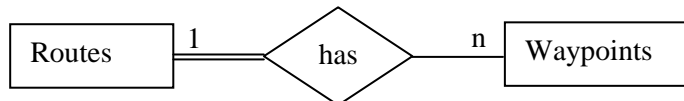
The server is based on JSP/Servlets/Beans and consists of 3 packages:

- Beans
- DB
- Routeservice

On the server there is also a database, which has 2 tables:

- Routes
- Waypoints

Among these 2 tables there is a 1:n relation: a route consists of many waypoints and a waypoint is just part of one single route.



3.2.1 Beans

There exist two beans: one for a route and another for a waypoint. A route consists of id, distance, altitude difference, speed, description, date, rating, grade, title and several waypoints. A waypoint has values for longitude, latitude and altitude.

3.2.2 DB

The database package contains a Connectionmanager, which allows us to connect to the database.

3.2.3 Routeservice

This is the main package of the server and contains all servlets, which are the interface between the client application and the web service. In this package there are also servlets which are used for the website (accessed by JSPs).

UploadRoute

This servlet is called by the clients, retrieves the following route information by POST-parameters and stores it in the database: distance, altitude difference, speed, description, title, rating, grade and date. Surely a route has also many waypoints. This information it requests by accessing the following parameters: count, lon<i>, lat<i>, alt<i>, time<i>, where <i> is a value from 1 to count.

SearchNearLocationServlet

This servlet is called by the clients when routes near the users' actual position are searched: using GET-parameters, the J2ME client passes longitude and latitude to the server who searches all waypoints in the neighborhood. There are returned id, title and distance of all routes, where at least one waypoint has a distance of less than 1 km to the actual position.

Routes are passed as text/html to client:

```
<routeid1>;<distance1>;<title1>\n
<routeid2>;<distance2>;<title2>\n
...
```

We used the “Haversine” formula to calculate circle distances between the two points – that is, the shortest distance over the earth’s surface:

```
R = earth's radius (mean radius = 6,371km)
Δlat = lat2- lat1
Δlong = long2- long1
a = sin2(Δlat/2) + cos(lat1).cos(lat2).sin2(Δlong/2)
c = 2.atan2(√a, √(1-a))
d = R.c
```

SearchByKeyword

This servlet is called by the clients when routes are searched by keyword: using GET-parameters, the J2ME client passes a keyword to the server who searches it in all routes descriptions and titles. There are returned id, title and distance of all routes which matched.

Routes are passed as text/html to client:

```
<routeid1>;<distance1>;<title1>\n
<routeid2>;<distance2>;<title2>\n
...
```

GetRoute

This servlet is called by the clients when they want to download a route: using GET-parameters, the J2ME client passes the routeid to the server who returns all information of that route as text/html:

```
<routeid>\n
<description>\n
<date>\n
<rating>\n
<grade>\n
<title>\n
<distance>\n
<lat>\n<lon>\n<alt>\n<time>\n           // Waypoint 1
<lat>\n<lon>\n<alt>\n<time>\n           // Waypoint 2
...
```

RoutesAll

This servlet is called by the website, running on the server. It retrieves all routes from the database and stores them into a list of RouteBeans. That list is passed using a “request attribute” and the “request dispatcher” to a JSP, which is responsible for displaying it as HTML to a user.

ShowMap

This servlet is called by the website, running on the server. It is responsible for retrieving all information of a single route (given by id) and all its waypoints. It inserts this information in a RouteBean and some Waypoint beans. Those beans are passed as “request attributes” and using the “request dispatcher” to a JSP, which is responsible for displaying the routes information as HTML.

4. Human/Computer interaction

4.1 Client-side interaction

A good canopied graphical user interface is especially important for mobile applications since there is only a small screen, no pointing device such as a mouse and only a few buttons with a restricted keyboard.

4.2.1 Home screen

The home or main screen is shown immediately when the midlet is launched. From this screen all further functionalities can be reached.

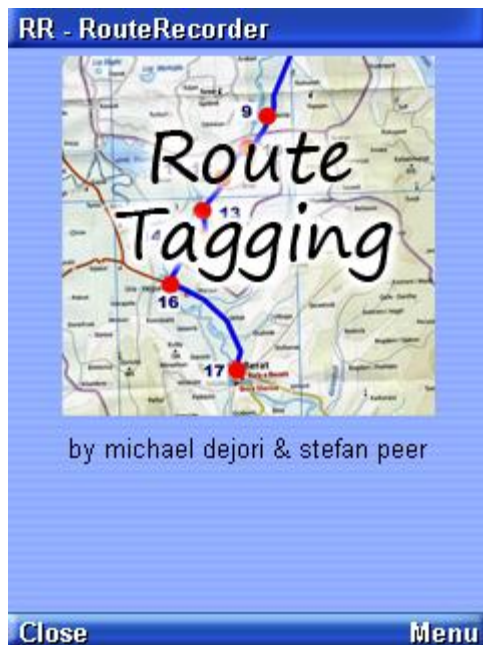


Figure 1: shoes the main screen

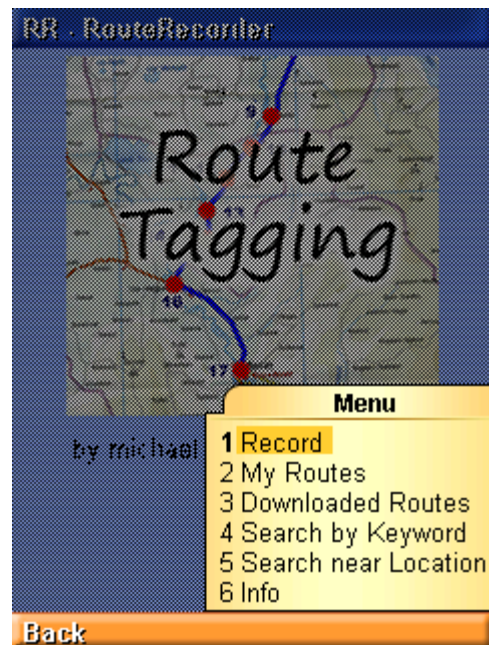


Figure 2: when pressing the menu button, there appear the menu points

4.2.2 Recording a new route

When we want to record a new route, the midlet switches to a new view, from which we can start recording.



Figure 3: empty route tagging form, where we can press Start Recording if we want to tag a new route



Figure 4: shows the screen after that four locations are retrieved from the location API. We can see the time which is elapsed and below there is the time and the street of the location snaps.

As soon as the user comes to the end of the route, he can press *stop recording*. As a sequence the last location will be retrieved at the end the summary of the route is shown.

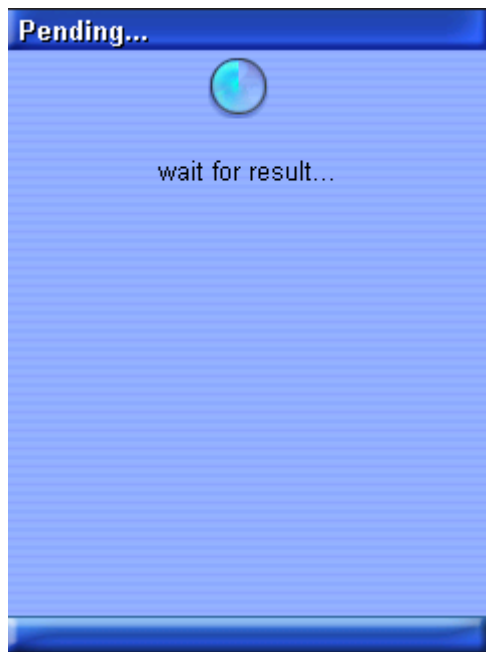


Figure 5: Meanwhile the last location is asked and the calculations about the summary are running, the pending screen is shown.

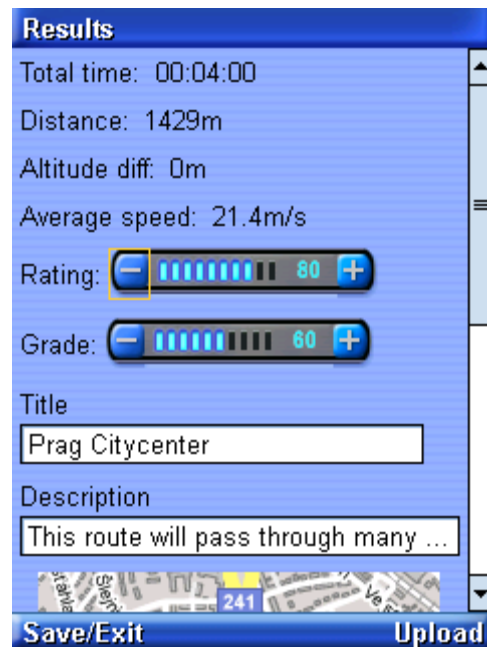


Figure 6: result screen; we can see the summary of the route. The duration of this route was 4 minutes, the distance 1429m. There is no altitude difference and the average speed was 21 m/s. Moreover the user can evaluate the route with a rating and difficulty. Additionally, he should give a title and description that is useful for other users.

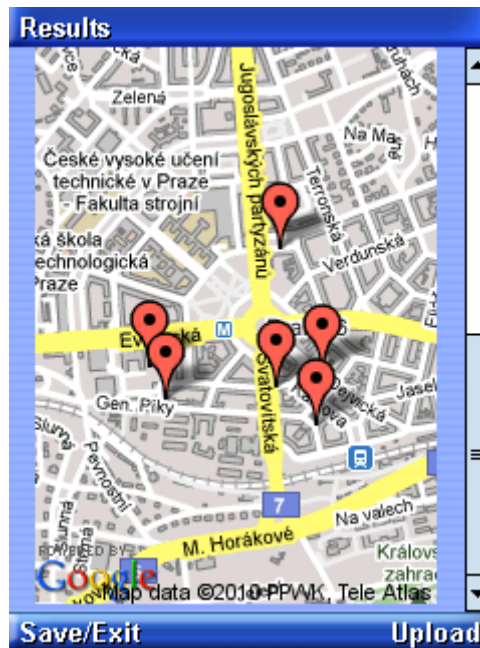


Figure 7: The user can see also a map with all waypoints which have been tagged.

As we can see from the above screen, the user has now two possibilities. He can either save the route locally on the mobile phone in a recordstore, where the route can be reconstructed and probably uploaded later or he can upload the route immediately to the web service. In this case it will not only be uploaded but also saved locally. After both operations the midlet will return to the main screen.

4.2.3 My Routes

When I want to see all routes that I tagged with my mobile phone I can just go to *MyRoutes*.



Figure 8: shows my routes. I can see the title and the date when I tagged it. Moreover we can see if the route is stored just locally (red) or if I uploaded it already (share it to other users).

From this list it is possible to delete a route from the local recordstore or view the details of a route.

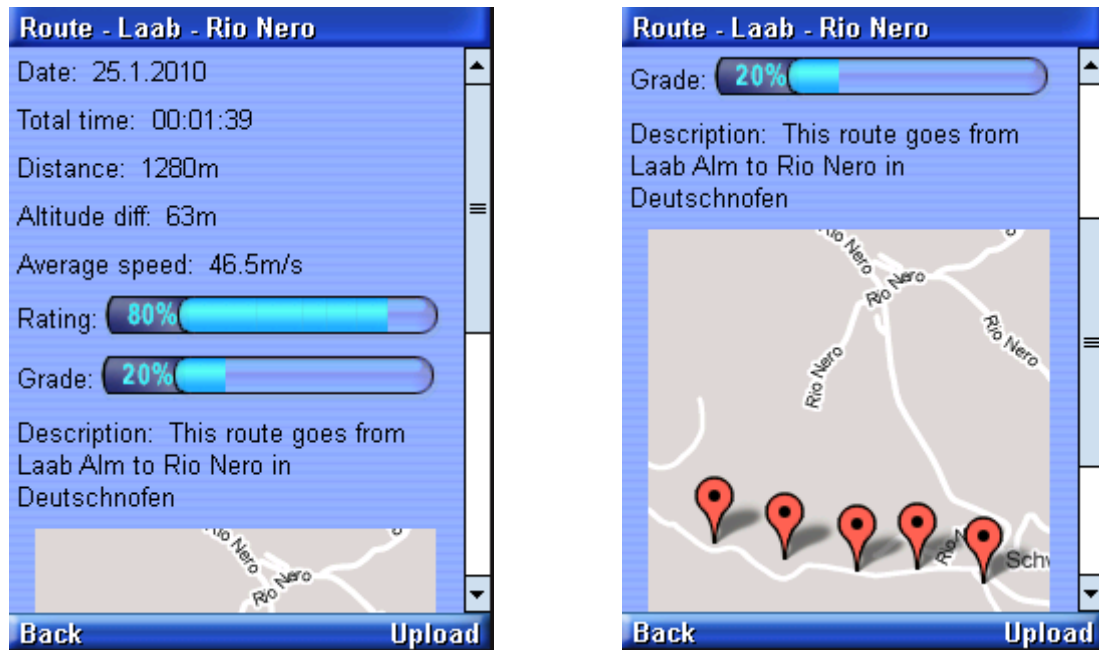


Figure 9: shows the details of one of the routes. I have also the possibility to upload it now if I didn't do it so far.



Figure 10: After uploading the route I get back to the list and I can see now that the route "Laab - Rio Nero" is already uploaded. When I select it again, I do not have anymore the possibility to upload it (deny uploading the same route more times).

4.2.4 Search by keyword



Figure 11: shows the form where I can specify the keyword it should be looked for.

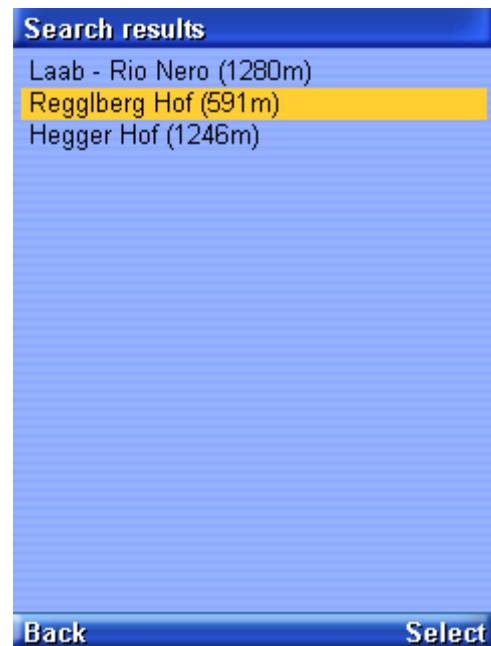


Figure 12: shows the results from the query.

The application will send the search term to a web-service and gets as response the list with the routes, which contain the keyword in description, title or in the address of one of the waypoints. If the user selects one route, the detailed information are requested from the web-service and he can save the route locally.

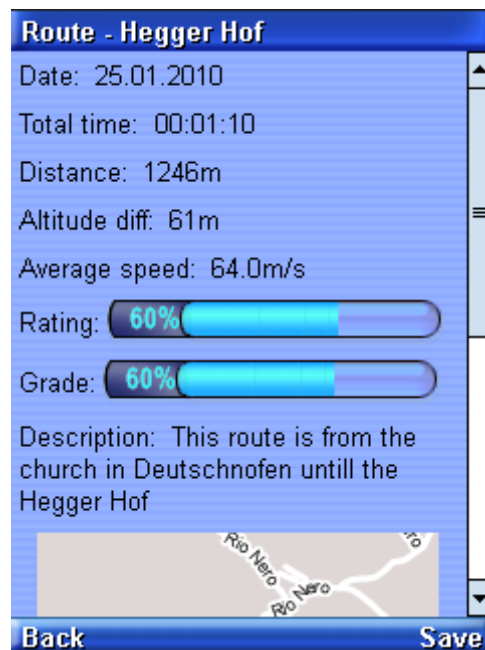


Figure 13: details of a shared route, with the map at the bottom.

4.2.4 Search near location

When the user is in a certain location and wants to see if in the neighborhood there is a walk way then he can just search for routes near his actual location.

He will get again a list with the results and can just proceed as before: selecting a route and then see the details and possibly save it locally.

4.2.5 Downloaded Routes

Under the menu *Downloaded Routes* I have all the routes that I downloaded from other users by using the search functionality. In this list I can select a route or even delete it from the *Downloaded Routes*.

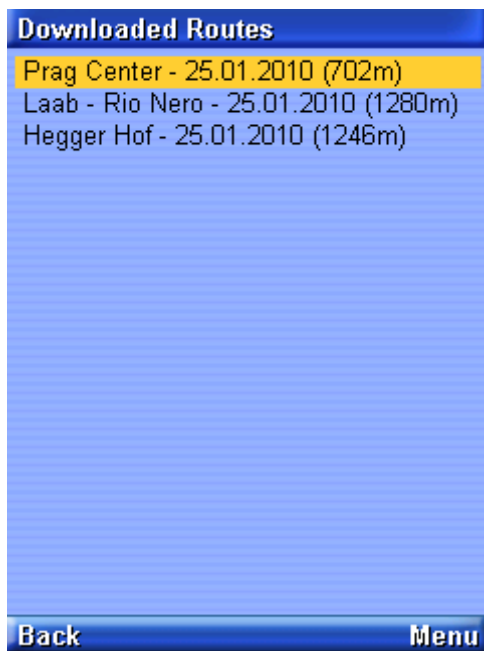


Figure 14: shows the downloaded routes

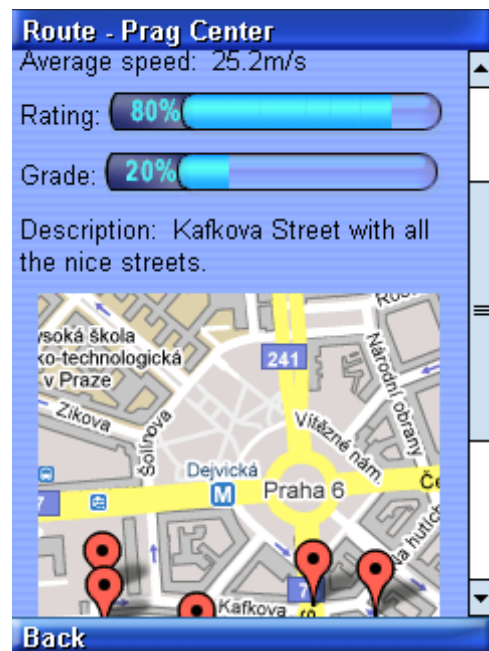


Figure 15: shows all details about a downloaded route

4.2 Server-side interaction

The information which is send to the server cannot only retrieved using the application, but also using a web-service which we developed. The service fetches all routes from the database and shows them in a list. Then a user can select a route and see a detailed map with the description queried from Google maps API.



- [1. Prag Center - 2010-01-25 \(distance: 702m, grade: 1, rating: 4\)](#)
- [2. Laab Alm - 2010-01-25 \(distance: 252m, grade: 4, rating: 5\)](#)
- [3. Laab - Rio Nero - 2010-01-25 \(distance: 1280m, grade: 1, rating: 4\)](#)
- [4. Reqqberg Hof - 2010-01-25 \(distance: 591m, grade: 3, rating: 4\)](#)
- [5. Heqger Hof - 2010-01-25 \(distance: 1246m, grade: 3, rating: 3\)](#)

© 2010 - MOBILE SERVICES - STEFAN PEER UND MICHAEL DEJORI

Figure 16: shows the list of routes in the database

route prag center

Distance of the route: 702
 Altitude difference: 0
 Speed average: 7.02 m/s

Description: Kafkova Street with all the nice streets.
 Date: 2010-01-25
 Rating: 4
 Grade: 1

A Kafkova

0,6 km (ca. 2 Minuten)

1. Richtung Nordwest auf Kafkova	32 m
2. Nach rechts abbiegen, um auf Kafkova zu bleiben	80 m
3. 1. rechts auf Dejvická nehmen	0,1 km
4. 1. rechts auf Kyjevská nehmen	76 m
5. Bei Kafkova rechts abbiegen	21 m
6. 1. links auf Wuchterlova nehmen	62 m
7. Nach rechts abbiegen, um auf Wuchterlova zu bleiben	84 m
8. 1. rechts auf Svatovítská nehmen	84 m
9. Bei Kafkova links abbiegen	18 m

Das Ziel befindet sich rechts

B Kafkova

0,3 km (ca. 1 Minute.)

1. West auf Kafkova Richtung Svatovítská	8 m
2. 1. links auf Svatovítská nehmen	80 m
3. 1. rechts auf Gen. Píky nehmen	0,2 km

C Gen. Píky

Figure 17: detail view of a route. The user has the possibility to zoom in the map or move the map. Moreover at the right side there is a detailed description from Google.

5. Google Maps API

5.1 KML

When a location is tagged, then as you have seen there is also shown the street name, where the user is.

This is realized by using a service which Google provides us. It is just necessary to request an URL with certain parameter and we get back a *kml* file with information.

<http://maps.google.com/maps?f=d&hl=en&saddr=<lon>,<lat>&daddr=<lon>,<lat>&ie=UTF8&om=0&output=kml>

The latitude and longitude is queried from the location API. After that the street is requested from Google by simply setting the start and destination address with these values. The parameter `output=kml` tells Google that I want a kml file as response.

Example:

```
http://maps.google.com/maps?f=d&hl=en&saddr=46.4062524820113,11.4389979905094&daddr=46.4062524820113,11.4389979905094&ie=UTF8&om=0&output=kml
```

Result maps.kml (shorted version):

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
  <Document><name>Daumstrasse to Daumstrasse</name>
    <Snippet><![CDATA[<a
href="http://maps.google.com/maps?f=d&hl=en&saddr=46.4062524820113,11.4389979905094&daddr=46.4062524820113,11.4389979905094&ie=UTF8&om=0">Printable
view</a>]]>
    </Snippet>
    <Placemark>
      <name>Head east on Daumstrasse toward Localit  Rio Nero</name>
      <address>Daumstrasse</address>
      <Point>
        <coordinates>11.438840,46.406390,0</coordinates>
      </Point>
      <LookAt>
        <longitude>11.438840</longitude>
        <latitude>46.406390</latitude>
        <range>100.000000</range>
      </LookAt>
    </Placemark>
    <Placemark>
      <name>Arrive at: Daumstrasse</name>
      <address>Daumstrasse</address>
      <Point>
        <coordinates>11.438840,46.406390,0</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

As we can see, we can parse the address *Daumstrasse* from the xml file by just sending the latitude and longitude.

5.2 Static image map

To retrieve a static image from Google it is just necessary to request the following URL

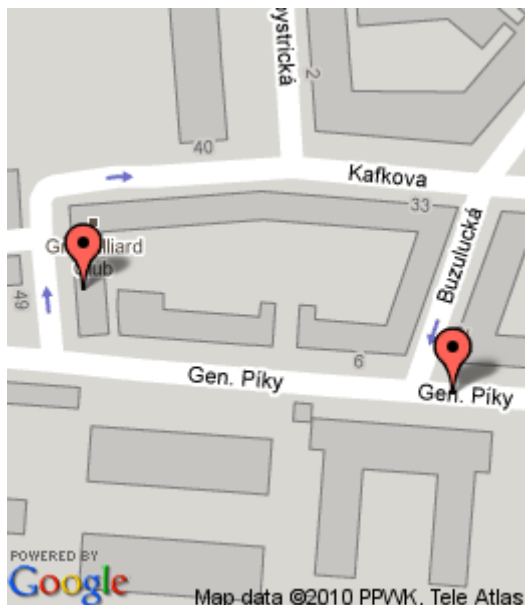
http://maps.google.com/staticmap?size=260x300&maptype=mobile&format=png32&markers=<lon>,<lat>&key=MAPS_API_KEY&sensor=false

I can specify different parameter, such as the size and format, which is important because I need a .png file to show the image in the midlet.

Example:

```
http://maps.google.com/staticmap?size=260x300&maptype=mobile&format=png32&markers=50.0982502049769,14.3931534532057|50.0986052742154,14.3911873044764&key=MAPS_API_KEY&sensor=false
```

Resulting image:



5.3 Include map and description in web service

To include a map of a route together with a direction description we can again simply use the Google Maps API.

We create a directions object and register a map and div to hold the resulting computed directions.

```
<script type="text/javascript">
  var map;
  var directionsPanel;
  var directions;

  function initialize() {
    map = new GMap2(document.getElementById("map_canvas"));
    directionsPanel = document.getElementById("route");
    directions = new GDirections(map, directionsPanel);
    directions.load("from: 46.4062524820113,11.4389979905094 to:
46.4055768635992,11.4384645039797 to: 46.4053551521908,11.4383875042068
```

```
to: 46.405133440799,11.4383105044397 to: 46.4044161468232,11.4379709674432  
to: 46.4042358399857,11.4378741568884");  
  
}  
</script>
```

The map will be shown in the div called *map_canvas* and the description of the direction in the div with id *route*. See figure 17 for instance.

6. Major technical problems and related solutions

We had some difficulties with the restricted version of the J2ME JDK. There exist many classes as Vector, StringBuffer, etc., but they don't provide the same methods as the standard JDK. Often we had to take a cumbersome way to reach our goal, because methods we normally used in the standard JDK were not present in the mobile version. If we take for example the "StringBuffer" class: in J2ME, this class doesn't provide the method "contains", which would often be very useful in comparing Strings.

Another problem which occurred was the problem with URL encoding: if we communicate from client to server (HTTP), all data has to be sent as parameters. If a parameter contains special characters or blanks, the communication fails. We provided a class URL encoder, which solves this problem.

Another difficulty we had was the planning of the project: we hadn't much experience and practice in J2ME programming, so it was not so easy to estimate the work load of the project.

7. Conclusion

We learned many things during this course. Previously we only heard about J2ME, but didn't know anything in detail about it. Now we made this project and are very happy with the result. It's another kind of programming, since you don't have such a big variety of functions like on other platforms.