UNIVERSITY OF ZAGREB
FACULTY OF ORGANIZATION AND INFORMATICS
VARAŽDIN

Marko Bartolić, mbartoli@foi.hr
Rene Škuljević, rskuljev@foi.hr
Tomislav Vunak, tvunak@foi.hr

NAVIGATION ON A SKI SLOPE

TECHNICAL DOCUMENTATION FOR SOFTWARE ANALYSIS AND DEVELOPMENT PROJECT

Varaždin, 2016.

UNIVERSITY OF ZAGREB
FACULTY OF ORGANIZATION AND INFORMATICS
VARAŽDIN

Team number: T10
Team members:
Marko Bartolić, 0016092146
Rene Škuljević, 0016092375
Tomislav Vunak, 0016091502

NAVIGATION ON A SKI SLOPE

PROJECT DOCUMENTATION FOR SOFTWARE ANALYSIS AND DEVELOPMENT PROJECT

Mentors:
Doc. dr. sc. Zlatko Stapić
Dr.Sc. Ivan Švogor
Evolaris mentors:
Gerald Binder
Thomas Rößler

Varaždin, 2016.

Table of contents

1 Introduction

1.1 Purpose of this document

The purpose of this document is to give a detailed description of the requirements for the navigation on a ski slope mobile software, called “EvoSki”. This document will explain the purpose and features of the application, interfaces of the application, what the application will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the application and will be proposed to the Evolaris company.

1.2 Intended Audience

The intended audiences are:

- Course mentors, to analyze the design and implementation of EvoSki app
- Evolaris mentors, to analyze the design and implementation of EvoSki app
- Authors of this document
- Eventual further developers

1.3 Scope

This document is intended to describe the design and technical specifications for the EvoSki application. Note that throughout the document, the “user” of the application will usually mean the person skiing on the ski track.

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	**Definition**
Recon Snow2	Ski goggles with integrated display.

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	**Definition**
API	Application Programming Interface.

2 General overview

2.1 Technologies used

EvoSki app is mobile application which is using Evolaris' webserver for fetching data about ski routes. While developing this application we used various technologies and tools for different activities.

Modelling tools (conceptual model, activity and class diagram, system architecture

- Visual Paradigm for activity diagrams

Version control system:

- Github repository available at: https://github.com/rskuljev/Navigation-on-a-ski-slope/

Web service and database:

- Used for testing purposes: FileZilla Client and MySQL database
- Used for application: web service given to us by Evolaris

Application development tools:

- Android Studio ver 1.4, Genymotion
- Microsoft Visual Studio 2010 (used for algorithm development)

2.2 General functioning

EvoSki application has the following functions:

- Obtaining user GPS coordinates
- Getting track coordinates
- Recognizing where user should turn next and how sharp the turn is
- Notifying the user where he should turn before approaching junction
- Showing the distance to the next turn and the distance to the end of the track
- Notifying the user if he deviates from track

2.3 Error handling

Application currently has error handling in the type of displaying messages if something unexpected happens. We made sure that the application does not crash, but instead just notifies the user what problem happened.

3 User requirements specification

3.1 Introduction

3.1.1 Objectives

This is the User Requirements Specification section for Navigation on a ski slope project, for use by Evolaris GmbH, team members and project mentors. In this section we will be defining the project's scope, user requirements that need to be satisfied by our application and describing our task assigned to us by Evolaris' mentors.

3.1.2 History

Evolaris already has a similar application developed and they would like to add some new functionalities which we were tasked to try and develop. Evolaris has sent us technical documentation and sample data on their existing web server so we can develop the fore mentioned functionalities.

3.1.3 Scope

While doing this project we are expected to create mobile application which will notify the user when he gets off slope, and to show him in which direction should he turn while approaching junction.

3.2 Organizational / Functional Areas Affected

3.4.1 Assumptions

The GPS data of the ski tracks that we use in the application are fetched from existing Evolaris web server, but we also used some of our own data for simulation.

3.5 Requirements

All requirements are defined below and are rated either Mandatory (M) or Highly Desirable (HD) or Desirable (D), dependent on business need and University Policy.

3.5.1 Functional Requirements

3.5.1.1 Common Features

Requirement	**Preference**
1.1.1.1. User can see the sharpness of the turn	M
1.1.1.2 User is notified when he leaves the ski track	HD
1.1.1.3 User is shown picture of turn when he gets near turn	D
1.1.1.4 User can set his own routes via txt file	D
1.1.1.5 User can send track start and end (GPS coordinates) to the web server, which return the track between these two points	D

3.5.1.2 Reporting

Requirement	**Preference**
1.1.1.6 Project documentation	M
1.1.1.7 Notes from SCRUM meetings	D

3.5.2 Production Requirements

3.5.2.1 Hardware

Requirement	**Preference**
Mobile device	M
1.1.1.8 Recon Snow2	D

3.5.2.2 Software

Requirement	Preference
1.1.1.9 Android min API 16	M

3.5.3 Development Requirements

3.5.3.1 Hardware

Requirement	Preference
1.1.1.10 Mobile device	M
1.1.1.11 Recon snow2	D

3.5.3.2 Software

Requirement	Preference
1.1.1.12 Android Studio	M
1.1.1.13 Genymotion	D
1.1.1.14 Microsoft Visual Studio	D
1.1.1.15 Evolaris Web Service	M
1.1.1.16 Retrofit library	D

4 Technical requirements

4.1 Evolaris Web server

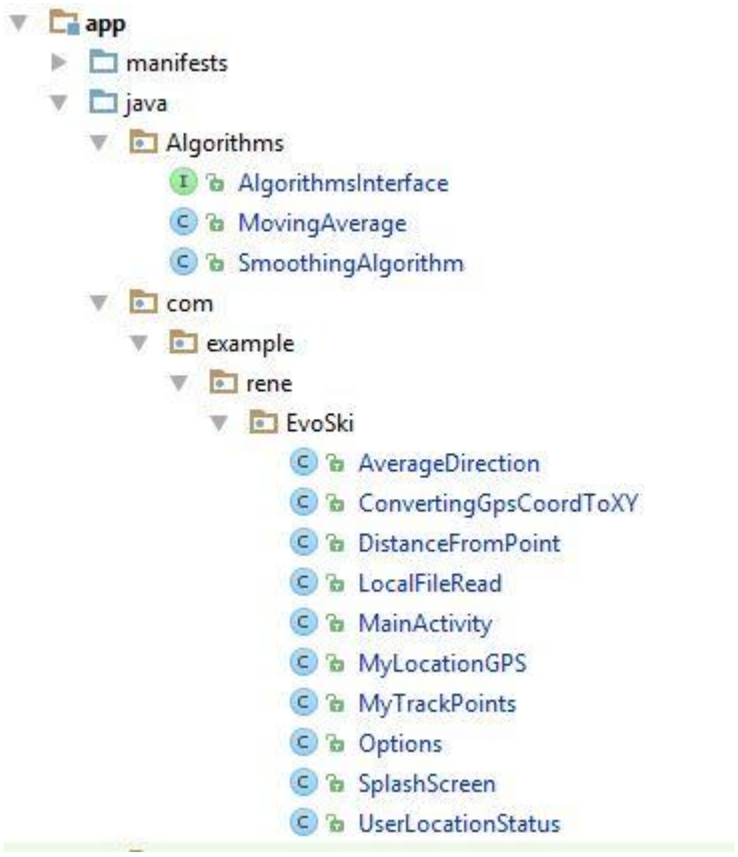
Evolaris provided us with access to their web server and data which we used for getting information about tracks. Application sends information about start and finish point through API to web service. The server calculates navigation from start to finish points and returns data in JSON format. It is important to note that the server doesn’t recognize road curves as turns on the track, but only intersections. Therefore, all the curves on the road are ignored and as such cannot be used as turns for a ski track.

As for the algorithm that checks whether the user left the track, the ideal scenario would be if the server returned track points which have the same distance between them as the width of the ski track. The algorithm works in a way that it compares the distance between two sequential points and the user’s distance to the next point. If the user’s distance to the next point is greater than the distance between these two sequential points, then application notifies the user that he is off the track. This would pretty good if we have small distances between two points, e.g. distance that is roughly the same as the width of the track. If the server returns points which are distant one from the other, like a road that doesn’t have intersection for a while, the application could notify the user that he is off track when he may not be.

5 EvoSki application source structure

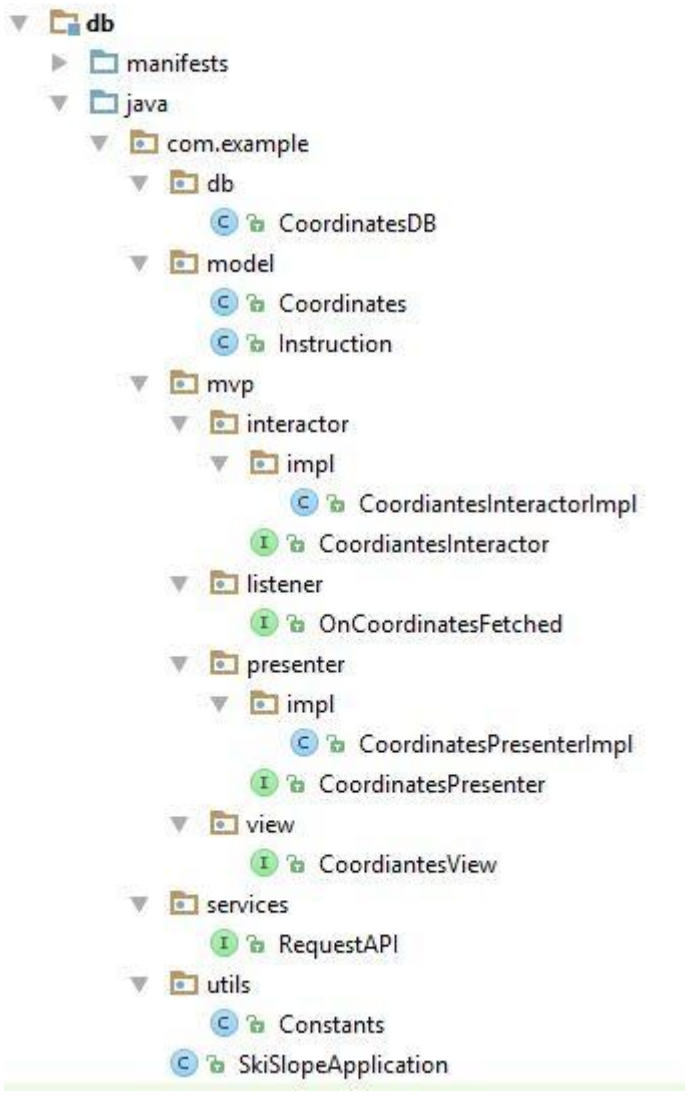
Application source code specification is generated automatically using JavaDoc, while we explained some parts of the code for easier understanding.

There are two main directories in the "Navigation on a ski slope" package – "app" and "db". "App" module contains the classes and activities that serve to communicate with the user, while the "db" module communicates with the web server and fetches track data.



```
app
  manifests
  java
    Algorithms
      AlgorithmsInterface
      MovingAverage
      SmoothingAlgorithm
    com
      example
        rene
          EvoSki
            AverageDirection
            ConvertingGpsCoordToXY
            DistanceFromPoint
            LocalFileRead
            MainActivity
            MyLocationGPS
            MyTrackPoints
            Options
            SplashScreen
            UserLocationStatus
```

Figure 1. App module



```
▼ db
  ▶ manifests
  ▼ java
    ▼ com.example
      ▼ db
          CoordinatesDB
      ▼ model
          Coordinates
          Instruction
      ▼ mvp
        ▼ interactor
          ▼ impl
              CoordiantesInteractorImpl
            CoordiantesInteractor
        ▼ listener
            OnCoordinatesFetched
        ▼ presenter
          ▼ impl
              CoordinatesPresenterImpl
            CoordinatesPresenter
        ▼ view
            CoordiantesView
      ▼ services
          RequestAPI
      ▼ utils
          Constants
        SkiSlopeApplication
```

Figure 2. DB module

6 Architecture

6.1 Conceptual application model

1. As user starts the application, he can currently choose between three types of track options.
2. Each track option represents a different type of getting the track information.
    a. “Fixed route” is a small track that goes through the streets of Varaždin. Coordinates and turns of this track are hardcoded in the source code of the application and that track was primarily used for our testing purposes.
    b. “Local file” is an option for loading track coordinates and turns from a txt file directly from the device. Currently, file name must be “TrackRoute”, and the file must be stored in default “Documents“ folder of the device (root\Documents). Contents of the file must be as follows: X coordinate of the GPS, then in the next line Y coordinate of the GPS, then in next line number which represents sign, i.e. sharpness of the turn (any positive values indicates right turn, and any negative value indicates left turn, e.g. -1 is left turn, 1 is right turn). That represents one point on track. Further points can be added right in the next line after that, by following the same rules. Obviously, if the file doesn’t exist, or the records inside are improperly defined, this option would not work.
    c. “Web server” is an option that allows the user to send track start and end points to the Evolaris’ web server, which then returns the whole track between these two points. Firstly, the “Options” dialog on the main menu must be opened. There the user can type the GPS coordinates of track start and end points. By pressing the “back” button, these coordinates are saved and by pressing “Web server” that coordinates are sent to the Evolaris’ web server which returns the whole track between that defined points. For this option to work, device has to have some type of internet connection and start and end coordinates of the track must be typed inside the “Options” screen.

Upon selecting any of these three options, the application calculates and shows the distance between the current user’s location and the next turn on track.

3. Application detects and monitors user’s movement pattern and locations. As he approaches the turn, i.e., when he is 30m away from the turn, the application shows the appropriate arrow which directs the user where he should turn, depending on user’s position in relation to turning point and the track itself.
4. At the same time, the application monitors user’s position and detect if he left the track. The algorithm works in a way that it compares the distance between two sequential points and the user’s distance to the next point. If the user’s distance to the next point is greater than the distance between these two sequential points, then application notifies the user that he is off the track. Ideal scenario would be if we have small distances between two points, e.g. distance that is roughly the same as the width of the track. If the server returns points which are distant one from the other, like a road that doesn’t have intersection for a while, the application could notify the user that he is off track when he may not be.

6.2 System architecture

System architecture that supports the EvoSki application consists of several elements. Communication between these elements is shown in Figure 3.

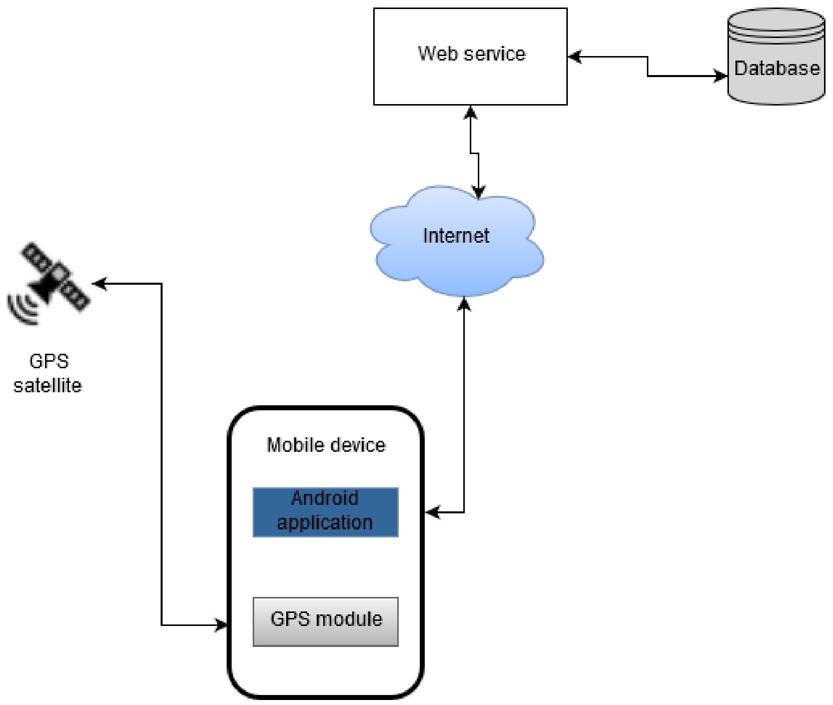


Figure 3. System architecture

Communication between architectural components:

1. Application requests GPS data from the GPS satellites.
2. Application connects to the internet and requests track data from the Evolaris’ web service.
3. Web service fetches track data from the database and sends it to the application.
4. Application also reads txt file from the device, if that option is chosen.

6.3 EvoSki application logic

On application startup, the user can choose between three track options. The first one, “Fixed route” has hardcoded coordinates in the Android application. The second one, “Local file” is by loading coordinates from a txt file. Lastly, third option, “Web server”, fetches track coordinates from Evolaris’ web server, where start and end points of the track must be typed in the “Options” menu. Furthermore, “Options” menu allows the selection of the desired algorithm that monitors user’s skiing and shows the appropriate arrow for turns. Currently it is possible to choose between “Real points”, “Smooth point algorithm” and “Moving average algorithm”. If neither option is selected, “Real points” remains as the default one.

Upon selecting any track option, the application checks whether GPS module on the device is enabled. If it is not enabled, the application notifies the user to enable it. Application constantly refreshes the user’s GPS location and shows distance to the next turn and total distance to track finish.

Next, the application communicates with backend server in order to fetch coordinates of the track and the appropriate turn locations. This communication is achieved through API provided by Evolaris. Application uses Retrofit to communicate with the web service which sends retrieved data from the database back to the application. Retrofit is a REST type client for Android, and it is built in a modular way and is part of Model-View-Presenter architecture. Next, that response is translated from the JSON object to the Java classes which the application uses to show the user his current GPS coordinates and remaining turns to the track finish.

Previously described logic, had to be modified for testing purposes. Because the distance between points, fetched from web service, was quite long. To make this easier, we created few track in Varaždin for testing purposes. Coordinates and turns of this tracks were hardcoded in Application.