UNIVERSITY OF ZAGREB

FACULTY OF ORGANIZATION AND INFORMATICS

V A R A Ž D I N

Nikola Smrček, nsmrcek@foi.hr

Josip Trupina, jtrupina@foi.hr

Božidar Labaš, blabas@foi.hr

**SKI LIFT WAITING TIME DISPLAY ON SMARTWATCH**

software analysis and development project

Varaždin, 2015.

UNIVERSITY OF ZAGREB

FACULTY OF ORGANIZATION AND INFORMATICS

V A R A Ž D I N

Team (T04): Bastion

Josip Trupina, full-time student

Nikola Smrček, full-time student

Božidar Labaš, full-time student

**SKI LIFT WAITING TIME DISPLAY ON SMARTWATCH**

software analysis and development project

Varaždin, January 2015.

Contents

[1. Introduction 2](#_Toc410306955)

[2. User Requirements Specification 3](#_Toc410306956)

[2.1. Introduction 3](#_Toc410306957)

[2.1.1. Objectives 3](#_Toc410306958)

[2.1.2. History 3](#_Toc410306959)

[2.1.3. Scope 3](#_Toc410306960)

[2.1.4. Issues 3](#_Toc410306961)

[2.1.5. Approach 4](#_Toc410306962)

[2.1.6. User Expectations 4](#_Toc410306963)

[2.2. Requirements 4](#_Toc410306964)

[2.2.1. Functional Requirements 4](#_Toc410306965)

[2.2.2. System Requirements 5](#_Toc410306966)

[3. Project development methodology 5](#_Toc410306967)

[3.1. Scrum team 6](#_Toc410306968)

[3.2. Product Backlog 6](#_Toc410306969)

[3.3. Sprint iterations 9](#_Toc410306970)

[3.4. Sprints 9](#_Toc410306971)

[3.4.1. Sprint 1 9](#_Toc410306972)

[3.4.2. Sprint 2 13](#_Toc410306973)

[3.4.3. Sprint 3 15](#_Toc410306974)

[3.5. The overview of sprints 16](#_Toc410306975)

[4. Technical documentation 20](#_Toc410306976)

[4.1. Conceptual application model 20](#_Toc410306977)

[4.2. Data model 20](#_Toc410306978)

[4.3. System architecture 21](#_Toc410306979)

[4.3.1. Google Cloud Messaging 21](#_Toc410306980)

[4.3.2. Communication between Smartphone and Smartwatch 23](#_Toc410306981)

[4.3.3. Web application 24](#_Toc410306982)

[4.4. Tools and technologies 27](#_Toc410306983)

[5. Ski lift waiting time display on smartwatch 28](#_Toc410306984)

# 1. Introduction

This is the documentation of the project from Software analysis and development course in collaboration with Evolaris GmbH company. The company was established in 2000 and since then they have continued growth. Their focus is on conception and development of applications for the web and mobile devices as well as the evaluation of business models for the economically useful application of mobile technologies. They are constantly searching for innovation projects in order to be more competitive and innovative. One of the innovation project is *ski lift waiting time display on smartwatch*.

The main goal of this project is to build an application on smartwatch that will indicate the current capacity off all ten lifts in the Schladming-Planai ski resort. The capacity represents values that variate from 0-100 in percentages. Indicator icon on smartwatch for all ten lifts will have exactly four different states expressed in four different colors. Green color represents low capacity of list, yellow represents medium capacity, orange represents high capacity. Red color will indicates that given lift is not functioning. The smartwatch is paired with a smartphone which acts as a gateway to the mobile network and provides access to backend server. The application is mainly for an arbitrary Android-Smartwatch (e.g. Samsung Gear 2 or higher). Users can watch the capacities of lifts in real time (data will be always refreshed), choose (filter) given lifts for display, change display of data and sort lifts.

At the end of introduction we would like to thank the Evolaris GmbH company for providing us the smartwatch which helped us a lot in developing process.



**Figure 1.** Sony Smartwatch 2

# 2. User Requirements Specification

## 2.1. Introduction

### 2.1.1. Objectives

This section covers the User Requirements Specification for *ski lift waiting time display on smartwatch* project that is mainly for project mentors and other members of team. This section contains project's scope and our approach, tasks determined by project mentors and requirements

from users that need to be fulfilled.

### 2.1.2. History

There is no current application or system that we use to continue working with our project. Our task was to develop completely new smartwatch application that will show capacities of lifts and web application that will run simulations for capacities of lifts.

### 2.1.3. Scope

Working on this project, we mainly focused on building a smartwatch application that will help users (skiers) who ski in Schladming-Planai ski resort to see the current state (capacity) of all lifts. Skiers can easily see capacities of all lifts in order to avoid those lifts with high capacity or those which are not functioning at the moment. We focused on data being constantly refreshed and many options for choosing given display which skiers like most.

### 2.1.4. Issues

While working on project we had an issues that is not critical, but relevant for mentioning it. Since, we are developing an application for smartwatch we could only test our results on one given smartwatch (Sony Smartwatch 2). Because of that we could not test our results (product) on other smartwatches that are newer and better and try some other features like wifi from smartwatch etc.

### 2.1.5. Approach

Our approach was mainly to build an application that can be easily tested by users and project mentors. Most important is that our focus for this project and this particular application is to provide members of Evolaris source code that is easy to understand and due its modularity easy to modify or add some new funcionalities, new GUI etc.

### 2.1.6. User Expectations

We want provide users an easy and practical way to see lift capacities. Users expectations are constantly refreshed data of capacities and adequate GUI. Users main goal is to use this application with minimal effort (for instance, not using smartphone for everything, GUI that is adequate in order to see data easy and clear). For project mentors and members of Evolaris company expectations include modularity of programming coude that is reusable and easy to understand/modify.

## 2.2. Requirements

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

### 2.2.1. Functional Requirements

#### 2.2.1.1. Common Features

| **Requirement** | **Preference** |
| --- | --- |
| * + - 1. User can see capacities of all lifts | M |
| * + - 1. User can change display (GUI) | HD |
| * + - 1. User can filter lifts | D |
| * + - 1. User can sort lifts | D |

#### 2.1.1.2. Reporting

| **Requirement** | **Preference** |
| --- | --- |
| * + - 1. Scrum meetings notes | D |

### 2.2.2. System Requirements

#### 2.2.2.1. Hardware

| **Requirement** | **Preference** |
| --- | --- |
| * + - 1. An arbitrary android-smartwatch | M |
| * + - 1. Mobile device with Bluetooth and version of Android OS higher than 4.0. | M |

#### 2.2.2.2. Sofware

| **Requirement** | **Preference** |
| --- | --- |
| * + - 1. Minimum Android API 14 | M |
| * + - 1. Eclipse | M |

# Project development methodology

We decided to use SCRUM methodology for this project development. Scrum is an iterative and incremental agile software development framework for managing product development.

Scrum emphasizes the idea of “empirical process control.” That is, Scrum uses the real-world progress of a project (not a best guess or uninformed forecast) to plan and schedule releases. In Scrum, projects are divided into succinct work cadences, known as sprints, which are typically one week, two weeks, or three weeks (possible four weeks or even more) in duration. At the end of each sprint, stakeholders and team members meet to assess the progress of a project and plan its next steps. This allows a project’s direction to be adjusted or reoriented based on completed work, not speculation or predictions. The tool we used for managing scrum methodology is QuickScrum which can be found on: <http://www.quickscrum.com/>.

## 3.1. Scrum team

The scrum teams consists of three different roles which are Product Owner, Scrum Master and Developer. Although, we strictly defined that all members of our team belong to one specific role (as shown in next Table 1.), everyone team member had contribution on developing some part of application.

|  |  |  |
| --- | --- | --- |
| TeamMember Name | Roles | Responsibilities |
| Evolaris mentors | Product Owner | Communication with clients,defining product backlog and goals. |
| Nikola Smrček | Scrum Master | Assigning tasks and resources, defining sprints and taskboard. |
| Božidar Labaš  Josip Trupina | Developer | Delivering incremented product after each sprint. |

**Table 1.** Scrum Team

## 3.2. Product Backlog

On the next table (Table 2.) we can see product backlog that we have defined. Product backlog is a prioritized feature list, containing short description of all functionality to be developed in product . There is list of nine elements to consider. First four of them are done in sprint 1 and next three are done in sprint 2 while last two elements are done in sprint 3. In order to see detailed information about every user story specifically see Table 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User Story | Type | Priority | Sprint name | Status |
| Establish communication between smartwatch and backend server via smartphone | Task | Major | Sprint 1 | Completed |
| Showing fresh updated data from server to smartwatch | Feature | Major | Sprint 1 | Completed |
| Web interface/service for simulations | Task | Minor | Sprint 1 | Completed |
| Indication about how fresh data is | Feature | Minor | Sprint 1 | Completed |
| Filtering lifts | Feature | Minor | Sprint 2 | Completed |
| Selecting type of displaying data | Feature | Major | Sprint 2 | Completed |
| GUI for smartphone | Task | Major | Sprint 2 | Completed |
| Modularize changing UI on smartwatch | Task | Major | Sprint 3 | Completed |
| Sorting lifts | Feature | Minor | Sprint 3 | Completed |

**Table 2.** Product Backlog

Here is detailed description about every user story defined in product backlog (shown in Table 3.)

|  |  |
| --- | --- |
| User story | Description |
| Establish communication between smartwatch and backend server via smartphone | It is very important to establish communication between smartwatch and backend server which runs simulations. Smartwatch is communicating with smarthwatch via Bluetooth wich acts like gateway..Smartphone communicates with server via mobile network or wifi. In order to establish this kind of communication we use Google Cloud Mesagging or shortened GCM. |
| Showing fresh updated data from server to smartwatch | It is important to users to have refreshed data on smartwatch as soon as server update data of simulations. |
| Web interface/service for simulations | The backend server simulates the real congestion data (values between 0% and 100%). In order to test results on smartwatch user can use simple user web interface to manipulate with simulations (e.g. turn on/off lifts, generate random number between 75-100 for particular lift). |
| Indication about how fresh data is | It is important for users to see time of last updated data. |
| Filtering lifts | User will have checklist in which he can select the lifts he wants in order to see data only about selected lifts. |
| Selecting type of displaying data | User will have an option to choose which type of display on smartwatch he wants. For instance, in shape of colours or as detailed lists. |
| GUI for smartphone | It is GUI on smartphone for users to use some settings for smartwatch aplication like choosing appropriate display or filtering options etc. |
| Modularize changing UI on smartwatch | It is important to modularize the concept of changing UI on smartwatch in order to easy adding new classes that will implement same interface for displaying data in different way. |
| Sorting lifts | User will have an option to choose what type of sorting he wants. The sorts will be from smallest to greatest, greatest to smaller and by alphabetical order of names of lifts. |

**Table 3**. Detailed description of user stories

## 3.3. Sprint iterations

We defined three sprints for our project development. We can see list of our sprints including the elements from product backlog that are within sprints and how long sprints lasted(see Table 4.).

|  |  |
| --- | --- |
| SPRINTS | SPRINT BACKLOG |
| Sprint 1 – completed (05/11/2014 – 14/11/2014) | Establish communication between smartwatch and backend server via smartphone |
| Showing fresh updated data from server to smartwatch |
| Web interface/service for simulations |
| Indication about how fresh data is |
| Sprint 2 – completed  (14/11/2014 – 13/01/2015) | Filtering lifts |
| Selecting type of displaying data |
| GUI for smartphone |
| Sprint 3 – completed  (13/01/2014 – 25/01/2015) | Modularize changing UI on smartwatch |
| Sorting lifts |

**Table 4.** Sprint iterations

## 3.4. Sprints

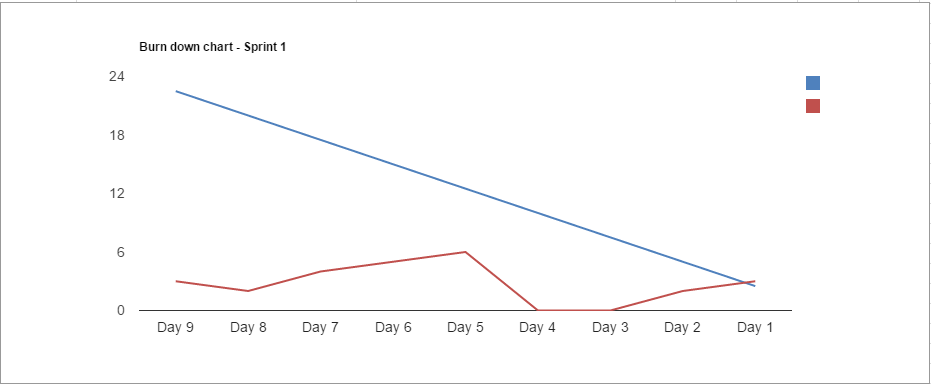
### 3.4.1. Sprint 1

The sprint 1 started on 5th of November 2014 and ended on 14th of November 2014. The result of sprint 1 and all tasks that have been done in sprint 1. First of all, we started by doing web part where we can simulate the capacity of ski lifts. The simulations run percentages between 0-100 and result are represented as chart. Also there is lift panel where we can manipulate simulation for instance, change lift's capacity from 75-100 to interval 0-25 etc. In order to see all tasks for sprint 1 see Table 5.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Product backlog Sprint 1 | Tasks | Resource | Estimation (hour) | Completed |
| Establish communication between smartwatch and backend server via smartphone | Establish communication between smartphone and GCM service | Božidar Labaš | 2 | 100% |
| Create database for GCM registration IDs | Nikola Smrček | 1 | 100% |
| Establish communication between smartwatch and smartphone | Božidar Labaš | 3 | 100% |
| Establish communication between server and GCM service | Nikola Smrček | 2 | 100% |
| Showing fresh updated data from server to smartwatch | Receive result of simulation from json format | Božidar Labaš | 2 | 100% |
| Web interface/service for simulations | Design web interface using mockup | Josip Trupina | 3 | 100% |
| Create and enable running simulations | Josip Trupina | 2 | 100% |
| Enable showing results of simulations as chart | Josip Trupina | 4 | 100% |
| Save results of simulations in json format | Josip Trupina | 1 | 100% |
| Create lift panel for manipulating data | Nikola Smrček | 4 | 100% |
| Indication about how fresh data is | Receive and show time of last updated data | Božidar Labaš | 1 | 100% |

**Table 5.** Sprint 1 – tasks

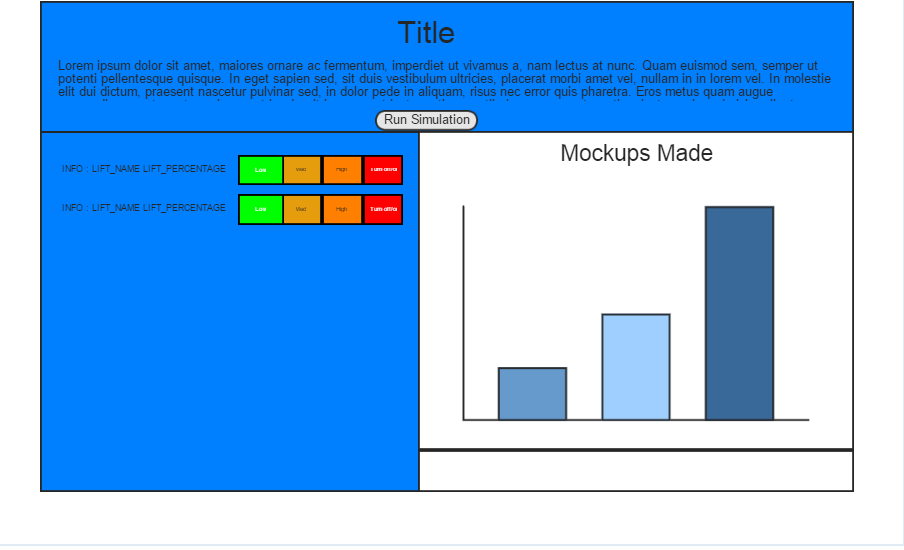
During sprint 1 we made some useful mockups to give as an idea how web simulation and smartwatch might look like (see Figures 3. and 4.). We also provide burndown charts for every sprint. Blue line represents ideal work and red line represents actual work. See Figure 2.



**Figure 2**. Burndown chart for sprint 1



**Figure 3**. Mockup for smartwatch



**Figure 4.** Mockup for web application

### 3.4.2. Sprint 2

The sprint 2 started on 14th of November 2014 and ended on 13th of January 2015. The result of sprint 2 and all tasks that have been done in sprint 2. First of all, we started by designing GUI for smartphone where user can change some settings of smartwatch application. User can select which lifts he wants in order to receive data only about those selected lifts. As required, user can change display of result data. For detailed information about tasks in sprint 2 see Table 6.

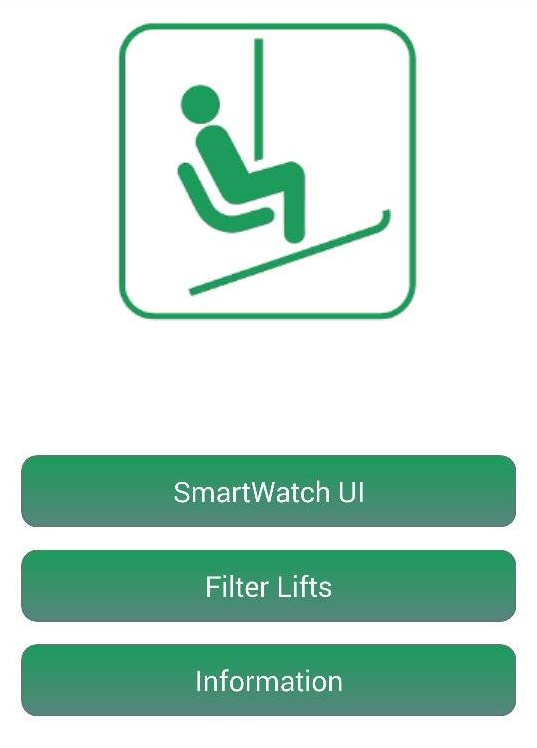
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Product backlog Sprint 2 | Tasks | Resource | Estimation (hour) | Completed |
| Filtering lifts | Design checklist | Božidar Labaš | 2 | 100% |
| Implement filtering | Božidar Labaš | 2 | 100% |
| Selecting type of displaying data | Create mockup for first type of display | Božidar Labaš | 1 | 100% |
| Create mockup for second type of display | Nikola Smrček | 2 | 100% |
| Design first display | Božidar Labaš | 1 | 100% |
| Design second display | Nikola Smrček | 2 | 100% |
| Implement upper tabs and slide effect | Božidar Labaš | 1 | 100% |
| GUI for smartphone | Create mockup for smartphone GUI | Josip Trupina | 2 | 100% |
| Design GUI | Josip Trupina | 2 | 100% |
| Implement functionalities of elements of the GUI | Josip Trupina | 2 | 100% |

**Table 6.** Sprint 2 - tasks

During sprint 2 we made some useful mockups to give as an idea how different displays of data on smartwatch (see Figures 5. and 6.). and smartphone GUI (see Figure 7.) might look like. In order to see burndown char for this sprint take a look at Figure 8.



**Figure 5.** Smartwatch display 1 **Figure 6.** Smartwatch display 2



**Figure 7**. Smartphone GUI

### b2.png

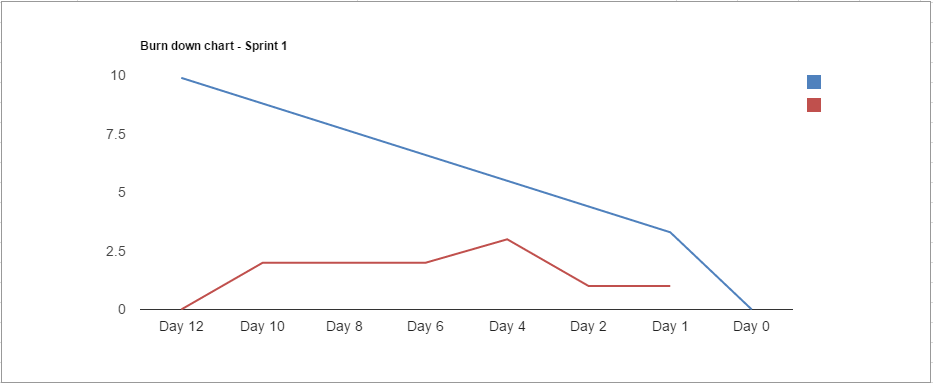
**Figure 8**. Burndown chart for sprint 2

### 3.4.3. Sprint 3

The sprint 3 started on 13th of January 2014 and ended on 25th of January 2015. The result of sprint 3 and all tasks that have been done in sprint 3. This sprint is all about two tasks. Our main focus was to increase modularity in a concept of changing user interface (display) on smartwatch in order to produce reusable code. Our focus to provide an easy way to implement our interfaces in order to add new classes that will take care of displaying data without changing anything else in the code. Also, we implement feature of sorting lifts. Also, we can see the balance between ideal and actual work on Figure 9 (burndown chart).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Product backlog Sprint 3 | Tasks | Resource | Estimation (hour) | Completed |
| Modularize changing UI on smartwatch | Implement classes for specific display on smartwatch | Božidar Labaš | 6 | 100% |
| Implement interface for changing UI | Božidar Labaš | 2 | 100% |
| Sorting lifts | Implement method for sorting lists | Božidar Labaš | 3 | 100% |

**Table 7.** Sprint 3 - tasks

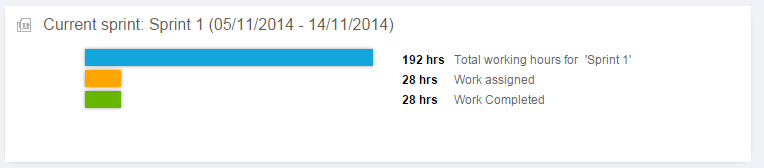


**Figure 9.** Burndown chart for sprint 3

## 3.5. The overview of sprints

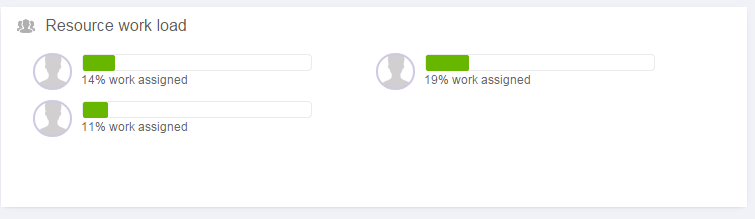
In this section we will show the total working hours predicted for all sprints. Also how many working hours were assigned and completed. There will be also a resource work load and task summary for every sprints

For sprint 1 we determined 192 working hours (period from 05/11/2014 to 14/11/2014 or ten days). 28 working hours were assigned and completed in sprint 1. See Figure 10.



**Figure 10.** Working hours for sprint 1

About resource work load 19% of work were assigned to Josip Trupina, 14% of work were assigned to Božidar Labaš and 11% of work were assigned to Nikola Smrček. See Figure 11.



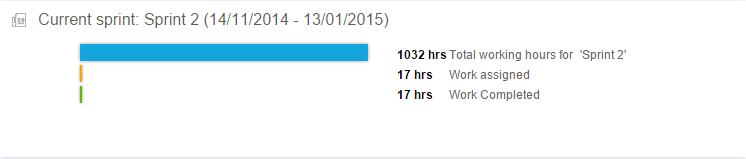
**Figure 11.** Resource work load for sprint 1

About task summary for sprint 1. Josip Trupina had five total tasks which are all completed in 15 hours of work. Nikola Smrček had three total tasks which are all completed in 7 hours of work. Božidar Labaš had five total tasks which are all completed in 9 hours of working. See Figure 12.



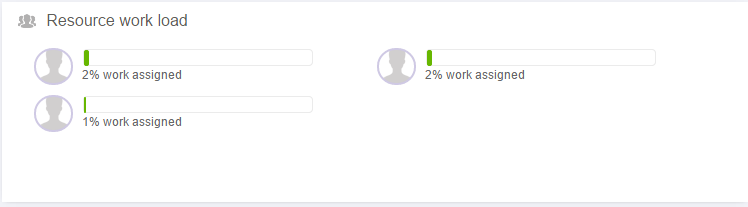
**Figure 12.** Task summary for sprint 1

For sprint 2 we determined 1032 working hours (period from 14/11/2014 to 13/01/2015). 17 working hours were assigned and completed in sprint 2. See Figure 13.



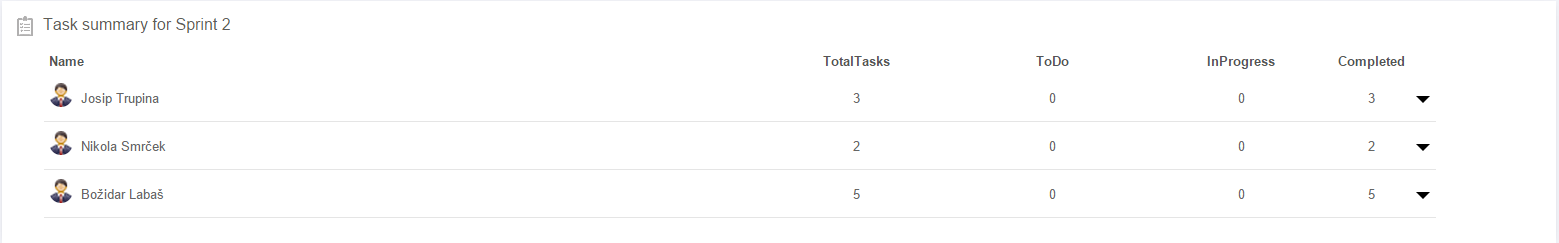
**Figure 13.** Working hours for sprint 2

About resource work load 2% of work were assigned to Josip Trupina and Božidar Labaš, and 1% of work were assigned to Nikola Smrček. See Figure 14.



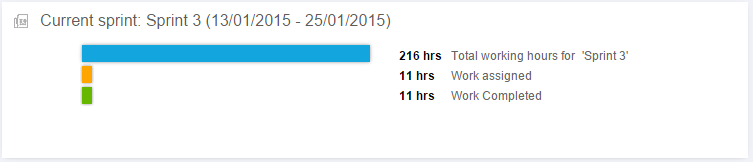
**Figure 14.** Resource work load for sprint 2

About task summary for sprint 2. Josip Trupina had three total tasks which are all completed in 6 hours of work. Nikola Smrček had two total tasks which are all completed in 4 hours of work. Božidar Labaš had five total tasks which are all completed in 7 hours of working. Here is task summary. See Figure 15.



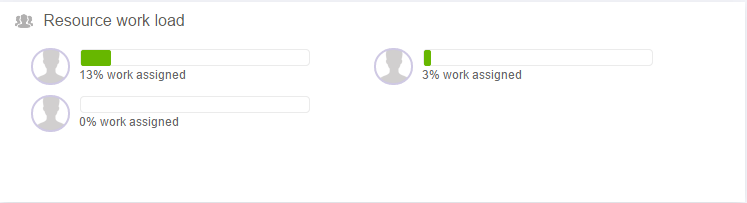
**Figure 15.** Task summary for Sprint 2

For sprint 3 we determined 216 working hours (period from 13/01/2015 to 25/01/2015 or twelve days). 11 working hours were assigned and completed in sprint 3. See Figure 16.



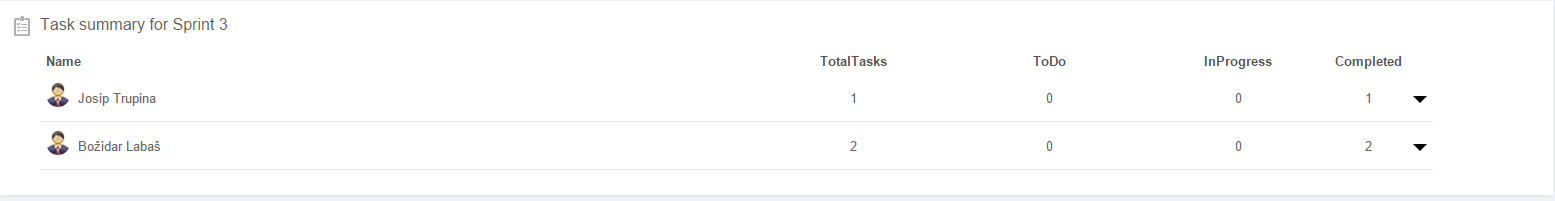
**Figure 16**. Working hours for sprint 3

About resource work load 3% of work were assigned to Josip Trupina, 12% of work were assigned to Božidar Labaš and 0% of work were assigned to Nikola Smrček. See Figure 17.



**Figure 17.** Resource work load for sprint 3

About task summary for sprint 3. Josip Trupina had one task which was completed in 3 hours of work. . Božidar Labaš had two total tasks which are all completed in 8 hours of working. See Figure 18.



**Figure 18.** Task summary for sprint 3

# Technical documentation

Tehnical documentation will introduce you to our application by conceptual application model which is similiar to user stories, data model, system architecture, tools and technologies.

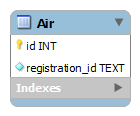
## Conceptual application model

This model defines possible activites which user can access. Their logical connection is shown below:

1. User starts application on smartphone where main menu pops. From the main menu he can choose to switch user design on smartwatch, filter lifts or information.
2. Choosing option „SmartWatch UI“ opens new activity where we can slide to select new user design on smartwatch, and confirming our selection by pressing „Select“ or pressing „Back“ to go back to the main menu.
3. Choosing from the main menu option „Filter lifts“ we see all the lifts with their names where by checking lift we are removing it from the the list on the smartwatch or clicking back on it to show it.
4. Choosing option „Information“ from the main menu opens up basic information about the application and also register's our smartphone to the database (for the GCM purposes).

## Data model

Database consistent of single table. Given table („Air“) is necessary for the functional operation of our application. Table function is to keep records of registrated smartphones. It has two columns (attributes) „id“ and „registration\_id“. Where „id“ is unique integer primary key (auto-increment) and „registration\_id“ type text any record must have both entries (attributes can't be null). Figure 19 represents described data model that consists of one table.



**Figure 19.** ERA model

## 4.3. System architecture

Our arhitecture consists of:

* Backend Server
* GCM Connection Server
* Smartphone
* Smartwatch

Google Cloud Messaging (GCM) for android is a service which is used to send the data from server to the devices with Android os. Smartphone and Smartwatch are communicating through bluetooth. On Figure 20. below we can see that communication betwen Smartphone and Smartwatch is one way communication. Smartphone gets data from server and pass this data to Smartwatch. Smartwatch do not send nothing to Smartphone.Our goal is to get updated data about ten Ski lifts capacity. This data is shown on Smartwatch.

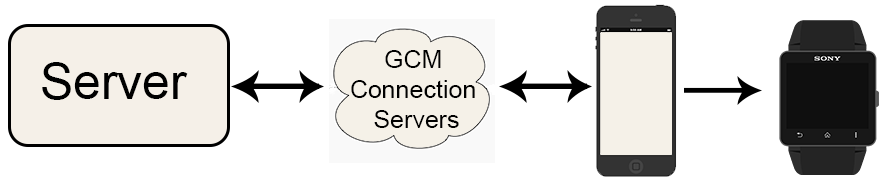


Figure 20. **System architecture**

### 4.3.1. Google Cloud Messaging

We can say that Google Cloud Messaging is „Man in the middle“ betwen our server and users device with Android os. GCM is a service that allows sending data from our server to users device with Android os, and also to receive messages from devices on the same connection. Most important functionality of GCM is handling all aspects of queueing of messages and delivery to the target Android application running on the target device, and it is completely free.

It is important to say how we implement GCM for our Application. The smartwatch is paired with a smartphone which acts as a gateway to the mobile network and provides access to our backend server. However, between smartphone and our server is GCM which we can describe like safety trigger in this case. See Figure 21.

Most important components of GCM communication:

1. **GCM connection server** : It receives the messages from our server and send these messages to the GCM enabled android devices.
2. **Our Application server**: It sends the message to the GCM connection server. We used PHP to build the application server.
3. **Android Application**: It receives the messages from GCM connection server after application server sends message to the GCM connection server.

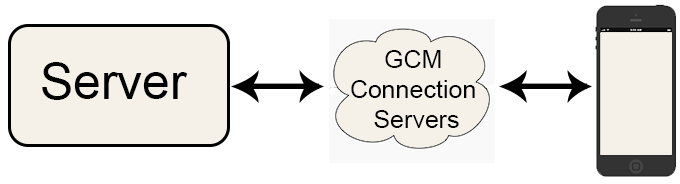


Figure 21. **Google cloud messaging**

Before smartphone get data from our server it needs to be registered on Google Cloud Messaging service. Our App on smartphone consists of user interface from where user can register on GCM. When user register to the GCM their Registration id is saved to our Database. Process of registration is described in six steps (see Figure 22.):

1. User can from our Android application enable the GCM by registering  to the GCM. The application needs Sender ID to get the registration ID. We have get Sender ID from Google services and put this id inside Application code.
2. GCM connection server receives the sender ID from application and returns the unique registration id.
3. The application send the registration ID to our application server for the storage.
4. The application server, stores the registration Id in the database.
5. When a new message need to send, the application server fetches the registration ids from database and send to the GCM connection serer along with the message.
6. The GCM server sends the message about Ski lift capacity to the application.

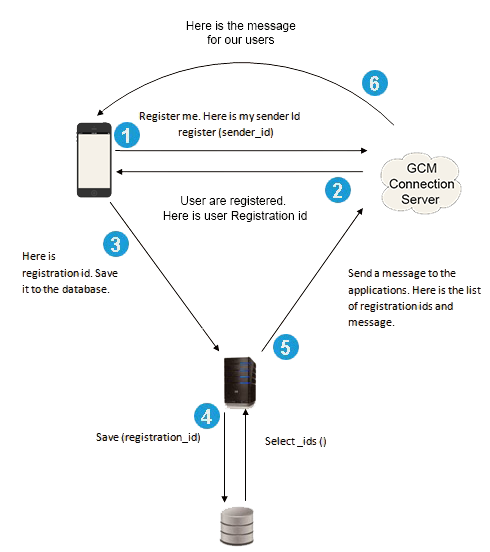


Figure 22. **Google cloud messaging register**

When users device is registered to GCM and their Registration id is saved to our Database we can establish communication betwen smartphone and smartwatch. On our server is php script with all required functionality for successfully communication. Backend server simulates the real congestion data (values between 0% and 100%).

### 4.3.2. Communication between Smartphone and Smartwatch

Smartphone and Smartwatch communicate with bluetooth. In our case Smartphone act as gateway to the mobile network and provide acces to out backend server from where Smartphone gets data about Ski lifts. Smartwatch is paired with Smartphone which only pass data to Smartwatch. With implemented GCM users can get updated data about Ski lifts. On picture below we can see that this is „one way“ communication. Smartwatch get dana from Smartphone, but Smartphone not not getting dana from Smartwatch.

More interesting is the GUI and interaction concept on the smartwatch. Data is shown on Smartwatch like list with ten Ski lifts.



Figure 23. **Communication between Smartphone and Smartwatch**

## 4.4. Tools and technologies

While working on this project we used many tools and technologies for our project development.

**Scrum methodology tools**

* QuickScrum which can be found on <http://www.quickscrum.com/>

**Modeling tools**

* Creately (for system architecture) which can be found on <http://creately.com/>

**Version control tools**

* Github

**Aplication development tools**

* Eclipse, PHPStorm IDE, NetBeans 8.0. IDE, GenyMotion

**Backend server and database**

* arka.foi.hr (server of Faculty of organization and informatics), PHP, MySQL. Access to web application on <http://arka.foi.hr/WebDiP/2013_projekti/WebDiP2013_079/>

**Technologies**

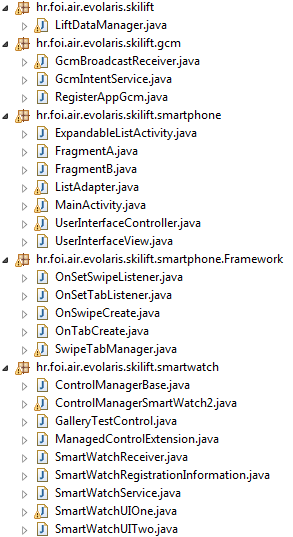
* GCM (see section 4.3.)

# 5. Ski lift waiting time display on smartwatch

## 5.1. SkiLift application

SkiLiftApp mobile application is built i Eclopse and at it has following functionalities:

* Register to Google Cloud Messaging on the first start of application. GCM registration id is stored on our database server and smartphone local memory. Registration process on GCM is only being done once on the smartphone application lifetime (until user uninstalls application)
* When lift data is changed on our service GCM push this data on user smartphone. User smartphone using broadcast receiver listen for gcm data.
* Smartphone is used as gateway between our server and smartwatch. When data is pushed to smartphone this data is via bluetooth forwarded to smartwatch.
* Smartwatch use lift data to draw user interface.
* From smartphone user can change smartwatch user interface, sort data and filter data which is displayed on smartwatch.



LiftDataManager – Singleton class which has maethods for store data, sort data and filter data.

GcmBroadcastReceiver - Receiver which listens for pushed data from GCM.

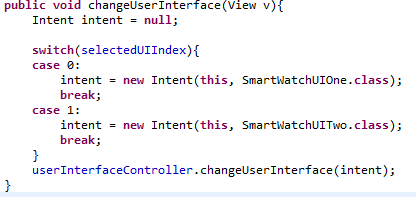
GcmIntentService – Service which is called from BroadcastReceiver. This service get data in form of intent and call LiftDataManager singleton class to store this data.

RegisterAppGcm – This class is used for registration on GCM.

## 5.2. User Interface module

OnUserInterfaceChanged interface and Lift class is exported to ski-lift.jar file. OnUserInterfaceChanged has one method drawUserInterface. Lift class is model for lift data. For now our application has two classes which is used to draw user interface on smartwatch. This two classes is SmarWatchUIOne and SmartWatchUITwo. This classes extends our super class ManagedControlExtension. This class is only helper class and extend super class ControlExtension. ControlExtension is class from sony library and its used to manage controls on smartwatch. ControlExtension is smartwatch class and its something similar to smartphone Activity. Our two classes implements OnUserInterfaceChanged interface and implements its method drawUserInterface.

For now we have two classes which draw user itnerface on smartwatch. This classes is SmartwatchUIOne and SmartWatchUITwo. When somebody wants to add new user interface him can simply create new class which extends ManagedControlExtension and implements OnUserInterfaceChanged interface. When its this new class („Control extension“) is called via intent. This is showed on picture below. UserInterfaceController singleton class manage and change smartwatch user interface.



When new smartwatch user interface class is implemented we need to add this change on smartphone so that user can select this user interface. For this purpose we implemented small framework which is used to simplify implementation of tab and slide images. Task of class UserInterfaceView is to extend this framework and show user collections of smartwatch designs. So, when programmer wants to add new tab and slide him only need to create new fragment which show smartwatch design and pass tab name to our setTab method.

## 5.3. Smartwatch GUI

Smartwatch GUI represents part of the application on smartphone where user can user some addition funcionalities like changing display, filterting lifts and sorting lifts by capacity. Initially when user starts application on smartphone it will load MainActivity. The next Figure 25. shows the main menu.

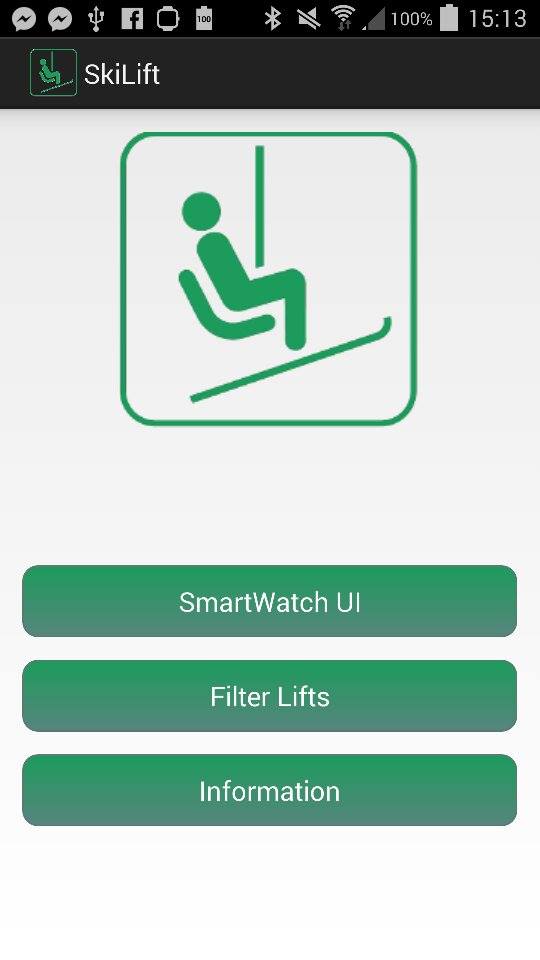


Figure 25. Main menu (MainActivity)

On the main menu user can select three options SmartWatch UI which loads new activity , Filter Lifts which loads new activity and Information button which shows a dialog with some information about this application. The next Figure 26. is activity that is loaded by clicking on button SmartWatch UI.

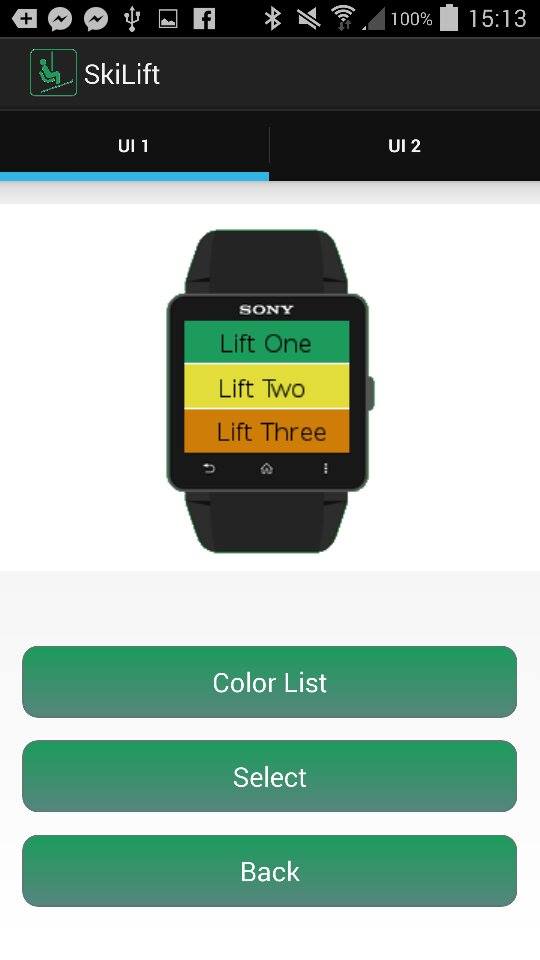


Figure 26. SmartWatch UI activity

This activity contains upper tabs in order to change which display user want. It works also on concept of swiping tabs. This activity contains fragments activity which are the UI1 and UI2. By pressing select it will instantly change display on smartwatch.

If user on the main menu clicks on Filter button it will load new activity which layout is shown in next Figure 27.

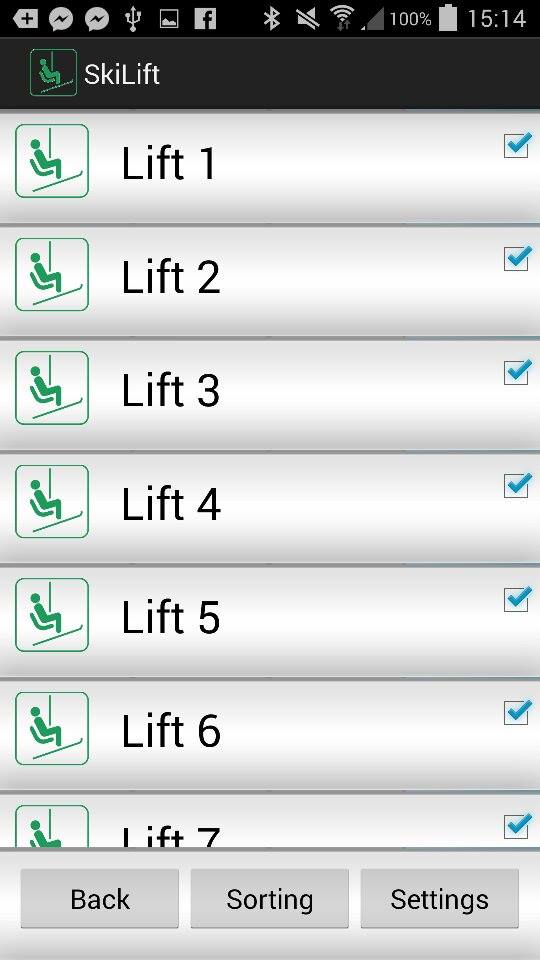


Figure 27. Filter activity

User can easily click on which lift he wants in order to turn off it's display of capacity of given lift on smartwatch. Also in this activity there is button Sorting which ask user wheter they want asceding or desceding sort. See Figure 28.

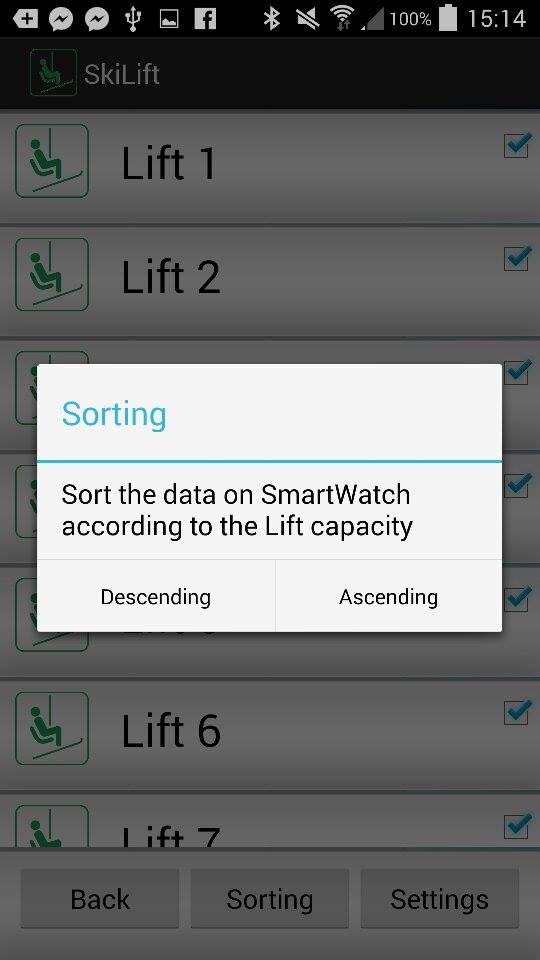


Figure 28. Sorting dialog

## 5.4. Web application

Web application is build as server side with php 5.5, and on the client side Javascript, HTML5, CSS3. Purpose of web aplication is some kind of control panel, so we will maybe call it „control panel“ in following text. Control panel itself is used to simulate behaviours of each created lift with buttons „Low“, „Med“, „High“ and „Turn off/on“. Each time we visit the web page we generate in „simulations.php“ ten lifts each with random percentages and status „1“. Following Figure 29. shows us the control panel for user.

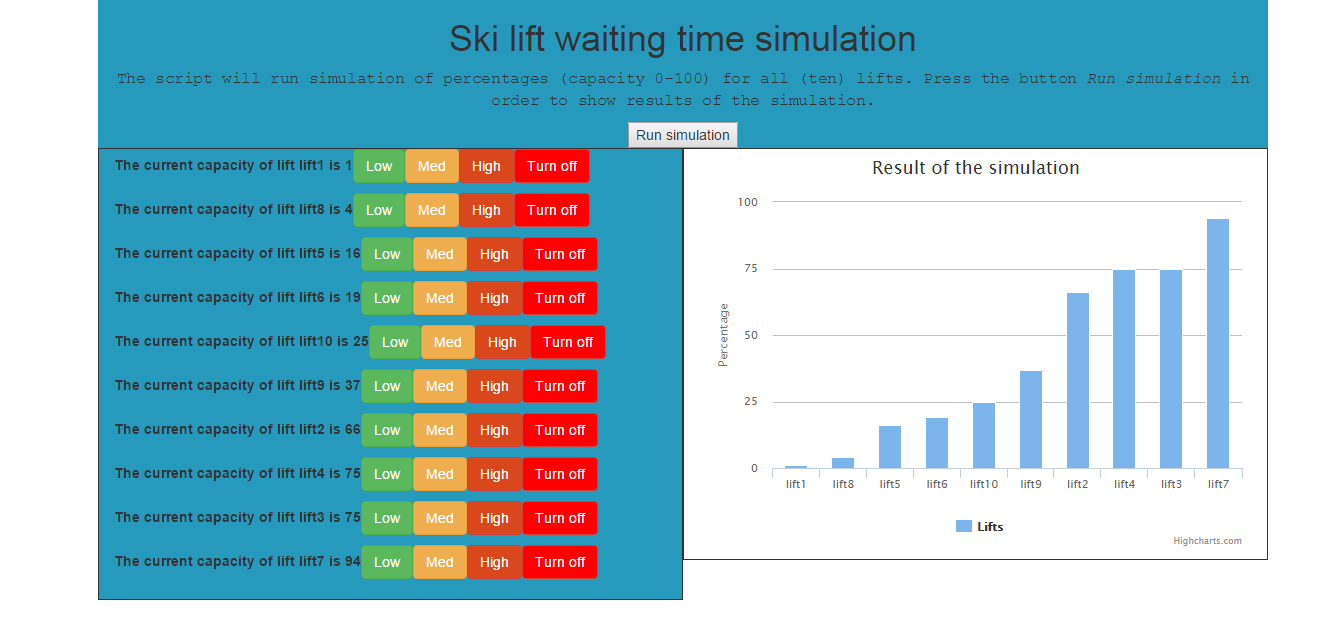


Figure 29. **Web application**

On the top of the application we have the it's title and description followed with button „Run Simulation“. On the left side we see list of all generated lifts and it's percentages with their control buttons. On the right side we see graphical view „result of the simulation“ displaying on the X-axis lift names, and on the Y-axis lift percentages.

In the web application we can differentiate three following cases:

1. User clicks button „Run simulation“
2. User clicks button „Low“, „Med“ or „High“
3. User clicks button „Turn off/on“

Web application current link : <http://arka.foi.hr/WebDiP/2013_projekti/WebDiP2013_079/>

As mentioned before we are using Javascript / Jquery with PHP. The application is single page, refreshing it's content without reloading page. We achieved that with Jquery, but the page it self is „index.php“. Javascript file „getJSON.js“ is doing the whole work with three arrays

* „status“ – status of each lift (0 = not in function, 1 = active /in function)
* „percentages“ – percentage value of each lift (200 = inactive –test purposes)
* „lifts“ – name of each lift

and three functions

* „refreshData(data)“ - takes parameter data which is content from „simulations.json“
* „createGraph()“ – fills graph with data from arrays „percentages“ and „lifts“ filtered by array „status“ (only active lifts can be displayed).
* „liftInfo()“ – fills the list with information from arrays „percentages“ and „lifts“ and control buttons filtered by array „status“
* „calculatePercentage(liftPercentage)“ – calculates new value for selected lift and returns it, function takes one argument so we can differentiate in which interval shall we generate data
* „switchStatus(index)“ – takes argument index of selected lift, so we can switch it's status

We splitted it into three different arrays beacuse it's much easier to fill highcharts data with data represented as seperated arrays then one array filled with object containing all information about it self. Information consistency is obtained from the fact that the each value of one lift is stored on same index, so iterating through one array is like iterating through all of them. Example from the picture, about lift on index place 2:

Lifts[2] value is lift5, status[2] value is 1, percentages[2] is 16

On the page load we read the data from „simulations.json“, fill the arrays in „getJSON.js“ which are used to manipulate the simulation. We dynamiclly create list of the lift with according information about it and buttons and also we create the graph object using „highcharts.js“ filling it with data of online lifts („status“=1).

**Case 1 – User clicks button „Run simulation“**

On the button click „Run simulation“ we are activating the button's eventListener. How it's inside „form“ HTML element it reloads the page. Web page „index.php“ is checking if some of the variables are set using POST or GET method. Using GET method we are using „simulations.php“ instancing it's object and running function „runSimulation()“. Which generates new values for lifts, sorts them and overwrites them to „simulations.json“.

Then activating functions in following order „refreshData(data)“, „createGraph()“, „liftInfo()“. So we can see new generated values for lifts and it's controls followed with graphical representation. At this point we are not yet sending any data to GCM (smartwatch).

**Case 2 – User clicks button „Low“, „Med“, „High“**

We are using same eventListener for all of those buttons so we are first checking which one did we click, upon that we are calling function „calculatePercentage(liftPercentage)“ where we can have argument „liftPercentage“ of values 1, 2 or 3. 1 represents that the new value shall be between zero and twenty-five (0-25), 2 (25-75), 3 (75-100). Using AJAX call we are transfering the data to server, so it can update new value to choosen lift. At this point on the server side we are calling „simulations.php“ and it's function „changePercentage“ which changes the according lift percentage value and saves it to „simulations.json“. Also we are send notification to GCM using function „sendNotification()“.Upon success callback we are updating values on the cilent side („percentages“ array, and calls to functions „createGraph()“ and „liftInfo()“).

**Case 3 – user clicks button „Turn off/on“**

Depending of the lift status it will be displayed turn off or on. The work around here is similar to the case 2. On click event we are changing the status of the selected lift, sending those days to the server („index.php“ using object instanced from „simulations.php“) so it can change it's status and update the data in the „simulations.json“, sending the notification to GCM. Upon the success callback to update the lift information, control buttons and the graph. We need to update graph beacuse we are only displaying „online“ lifts. On error callback (unsuccessful changes) nothing happens.