MÄLARDALENS HÖGSKOLA

Real-Time Bridge Monitoring
Design Description

Version 1.3

Revision History

Date	Version	Description	Author
2013-10-31	0.1	Initial Draft	Jörn Tillmanns
2013-11-04	1.0	First version of Design	Andrea Bottoli, Lorenzo Pagliari, Marko Brčić, Ghazal Shojaee, Jörn Tillmanns
2013-11-17	1.1	Fixed Introduction Fixed 2.1 Section Fixed some formats	Andrea Bottoli, Lorenzo Pagliari
2013-11-18	1.2	Fixed Sections 2.2, 2.3 Fixed Chapter 3	Andrea Bottoli
2013-11-19	1.3	Fixed User Interfaces Section	Andrea Bottoli

Table of Contents

1. Introduction

Real-Time Bridge Monitoring is a project for the Distributed Software Development course held by Politecnico di Milano, Malardalen University and University of Zagreb.

1.1 Purpose of this document

The purpose of this document is to assist in the development of the project 'Real-Time Bridge Monitoring", as a part of the Distributed Software Development course. This document specifies the entire architecture and design of the "RTBM"-Software. These design decisions directly relate to the requirements, use-cases, attributes and interfaces of the system, as they are mentioned in the Software Requirements Specification document.

1.2 Intended Audience

This document is written mainly for the development team. It will assist the team during the development-phase and will be updated, if any design-decision will be changed.

It'll be shown the software architecture

1.3 Scope

The scope of this document it to present the overall architecture, then go into the details of each model, interface, technology that we used and other architectural aspects needed.

This document aims to present much detail as possible with the software architecture of the entire system, including the database, the parser, the math engine and the web application; there are also specified in detail the external interfaces, on the one hand towards the 'end-user and the other to the hardware components.

You will then be presented in detail on the model chosen for implementation, including sequence and class diagrams, data flow, types and formats of data, the database model, including its tables and, finally, the organization of the website.

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definitions
Real-Time Bridge Monitoring	Project Title
Web App	The web application of the project

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
HTML	HyperText Markup Language
GUI	Graphical User Interface
AJAX	Asynchronous Javascript And XML
JSON	JavaScript Object Notation
MVC	Model View Controller
DB	Database
RTBM	Real-Time Bridge Monitoring

1.5 References

The main reference to this project is:

www.fer.unizg.hr/rasip/dsd/projects/real-time_bridge_monitoring

Other useful references:

- Project Plan: http://www.fer.unizg.hr/_download/repository/Project_Plan_v1.1.pdf
- Design Document: http://www.fer.unizg.hr/_download/repository/Design_Description_v1.0.pdf
- Requirements Document: http://www.fer.unizg.hr/_download/repository/Requirements_Definitionv0.01.pdf

2. Software architecture

2.1 Conceptual design

The system will be developed following the MVC pattern and will have some modules that will interfaces with each others to gather, elaborate, validate and present them data to the end-user.

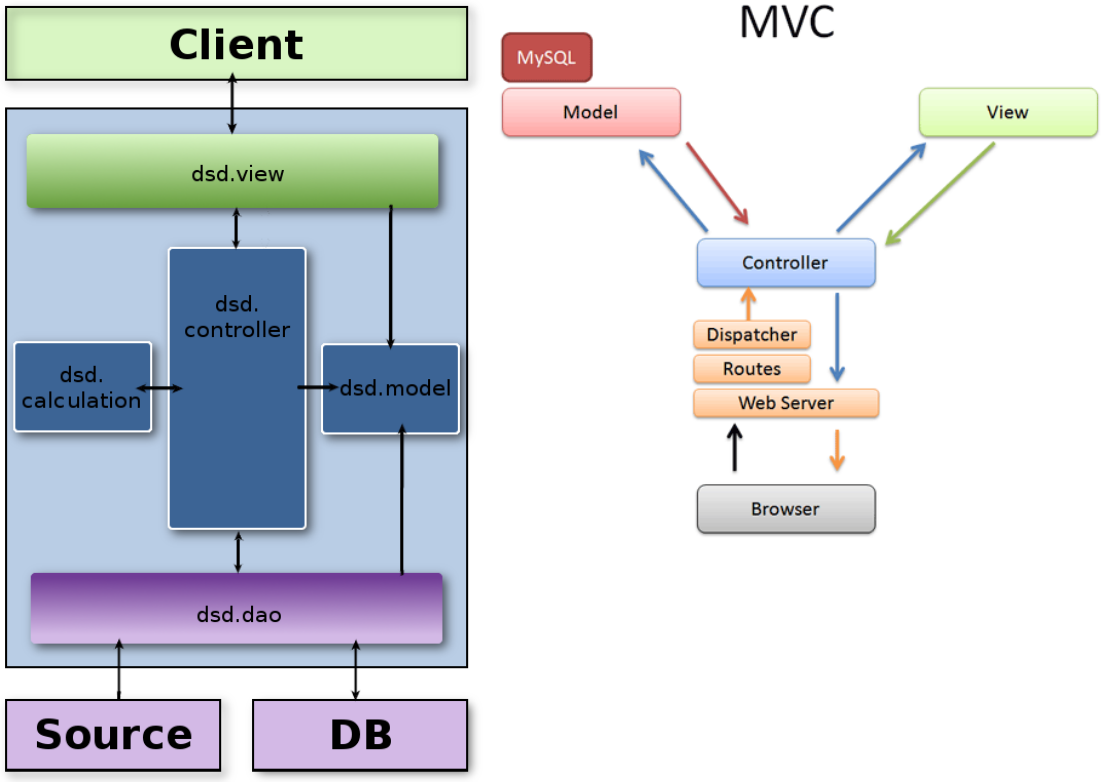
The dsd.dao component will provide a unique access to the data for the system, that could call it to retrieve the data.

The dsd.view will present to the final user the data concerned the system, the sensors and other stuffs needed to manage the real-time situation of the bridge.

The dsd.model will present to the other components the high level model of the data in the Database, like raw data, parameters, users, etc.

The dsd.calculation is the math engine of the system, and it calculate all the formulas for retrieving the safety factor of the bridge; plus, it fills some more aggregate data fields in the database according to the formulas and the structural calculations done.

The dsd.controller is the "logic" of the system, that controls the input by the user, manage the callings done by the subsystems and so on.



2.1.1 Source

It represent the input files that come from the sensors; they are two .jpeg files (picture taken from the camera on the bridge) and two .txt files (in which we can retrieve the timestamps and the data of the sensors).

2.1.2 Database

As Database we will use MySQL; it will, except of the media-data like photos and videos, contain all necessary data of the software. These data are:

1. the raw-data from the sensors
2. the pre-calculated sensor and danger-level data
3. user-data
4. parameters
5. technical instruments

2.1.3 dsd.dao

It is a layer between logic which is the upper level and storage which is the down side which is directly connected to source and db layer. This inherits from source and DB interfaces.

2.1.4 dsd.calculation

The data from the sensors are delivered as raw text-files; these txt files have one data per second for each sensor on the bridge (one Hydrometer, one Ecosounder and one Sonar) and they have to be parsed and written into the database (the local server on the bridge, which is not under out control, sends one package per hour that contains two txt files and two jpeg files, so we will have 3600 single data for each sensor).

During this process the script will detect and compensate inconsistent data; this script will be automatically running in certain intervals. We have also to archive images from two cameras and create a sort of movie from the daily, weekly and monthly pictures.

The script will be divided into four submodules:

1. The parsing-submodule will parse the data-files
2. The image-submodule will recognize the new images
3. The consistence-check will validate the read data and patch inconsistency (by the way, some other checkes will be done later in the system)
4. The SQL-submodule will write all data into the database
5. The film-submodule will create overview-movie for every day, week and month.

This component requires access to the data-files and the database.

The “calculation library” is the main-part of the software; it permits to the calculation engine to perform the structural calculations providing the formulas needed.

It will provide also the formulas to perform the calculations of the danger-level and other additional parameters based on the raw-data from the database.

The calculations will be performed in certain time-intervals and the calculated results will be stored in the database.

2.1.5 dsd.controller

This module controls the flow of data coming from the end-user, from the calculation library, from the data sources and from the database; it is the "logic" of the system and controls the correct behavior of the product.

Moreover it checks some validation of user input for instance log in.

2.1.6 dsd.view

The web service will present all data to the user and it is the only way how the user interact with the system.

To use the web service the user will use a common browser, the web service give him the possibility to log in and see information based on his access level.

Also for high access level there is the possibility to do some extra actions, as add or remove some users, give or remove access level, change parameters of calculation engine and so on. We will use a MVC pattern to develop the web service.

2.2 System specification

We use a Linux Debian 6.0 to build the server. To client side, the access to the web service can do by using each common browser as Google Chrome, Mozilla Firefox, Safari, Internet Explorer, Opera.

To programming we will use at server side Java for the calculation engine and the parser, and client side we will use html/css with JavaScript.

2.3 External Components

For the server we will use Apache with Tomcat for publish the web service, a daemon created by us as job scheduler to schedule the parser and calculation operations and MySQL to build the database.

On client-side we will use a simple and generic browser for display graphs.

3. External interfaces

3.1 Hardware Interfaces

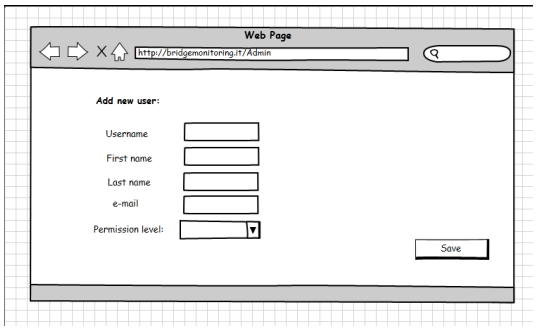
The system gathers values from the following sensors and cameras:

- Anemometer: to measure wind speed and direction
- Hydrometer: to measure the water height
- Echo sonar: to measure the height of the bottom of the river and its changes
- Camera: two camera that periodically take a photo of the bridge

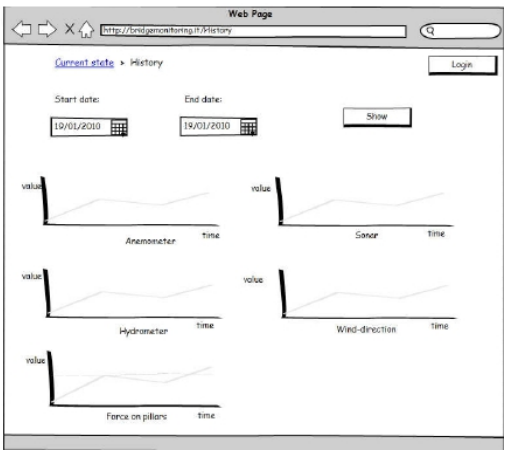
This hardware part is out of our control and we can't manage directly the sensors, the camera and also the way which the data is gather and send to the system; thus, we assume that we receive one package per hour composed by two txt files with raw sensor data, and two picture of the bridge taken by the camera.

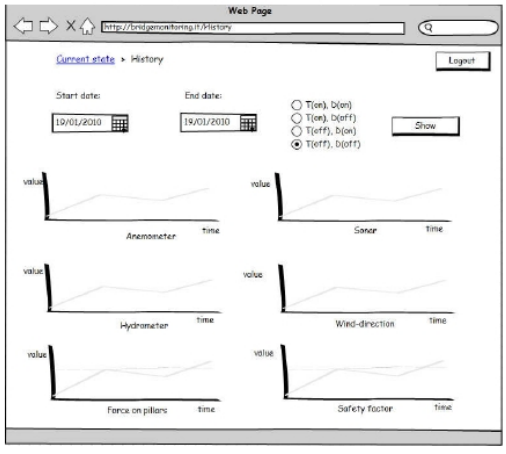
3.2 User Interfaces

3.2.1 AddUser



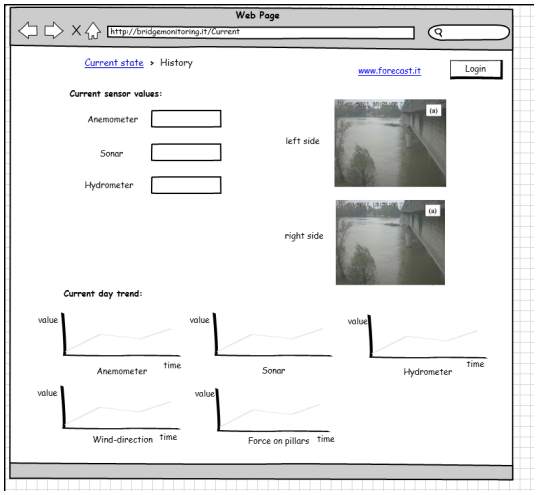
3.2.2 Historical View

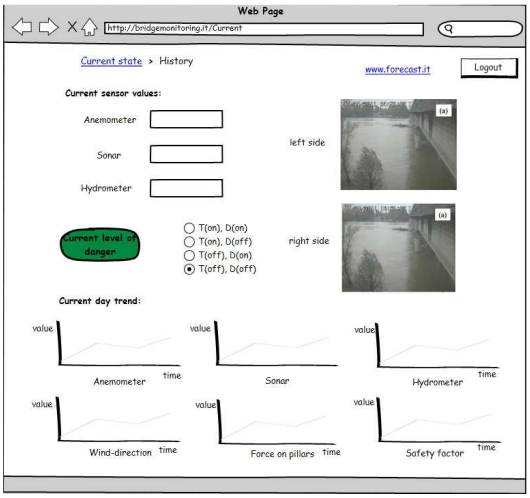




If the user is logged in, he will see also the risk-level (-diagrams):

3.2.3 *DataView*



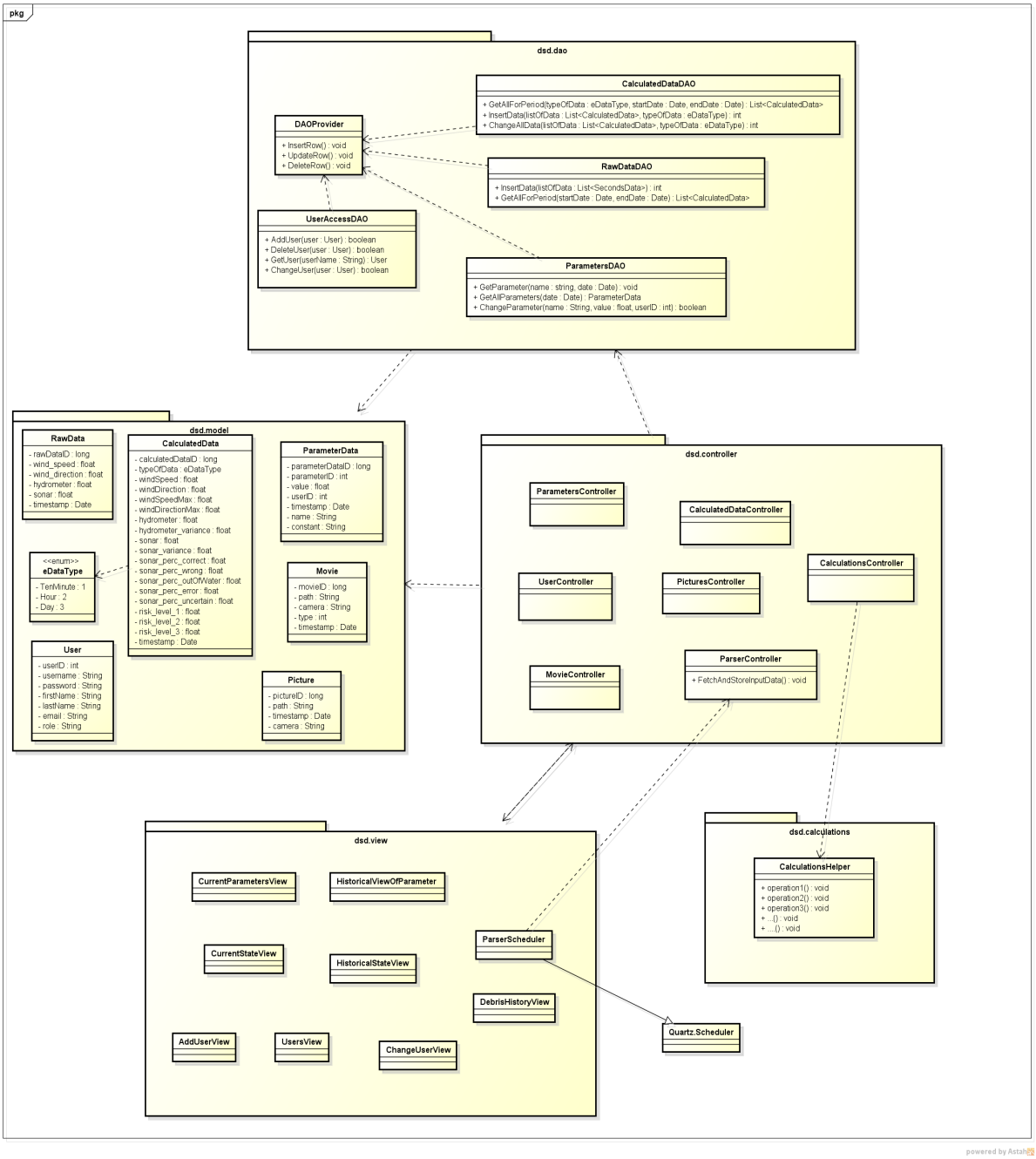


If the user is logged in, he will see also the risk-level (-diagrams):

4. Detailed software design

4.1 Implementation modules / components

4.1.1 Structured view

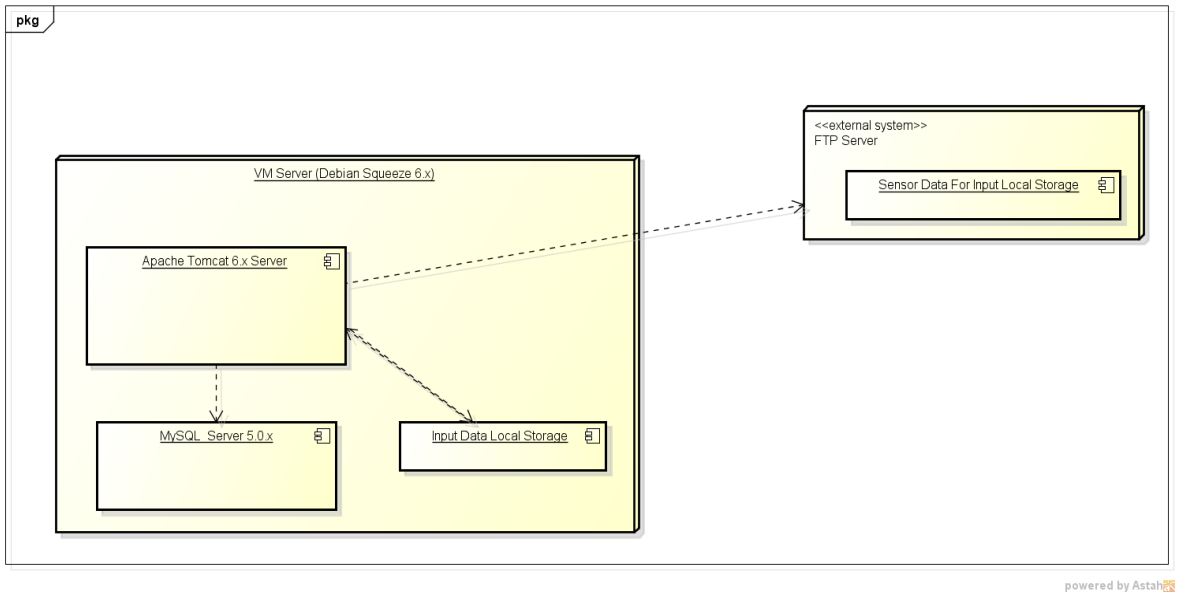


We are conforming to model – view – controller pattern. With additional layer for communicating with database called DAO. Since it is hard to plan the whole system design in advance, we were able to sketch the detail only for lower layers, since we have also the database design. When the development process will be at a further stages, we will be able to get the more detailed picture of the class diagram.

Layers described:

- DAO – layer for communicating with the database, knows about the controller and the model layer
- model – contains raw classes with their fields and properties
- controller – layer which combines all the other layer because other layers usually communicate through this layer
- view – layer that consists mainly from Servlets and .jsp pages

4.1.2 Deployment



The communication with the external server that holds the input files is still subject of discussion. What we received from the customer for testing are row files from the years 2011 and 2012. Because of that and because of the fact that we are not allowed to access the real system, we don't have the possibility to test the system for a real environment use.

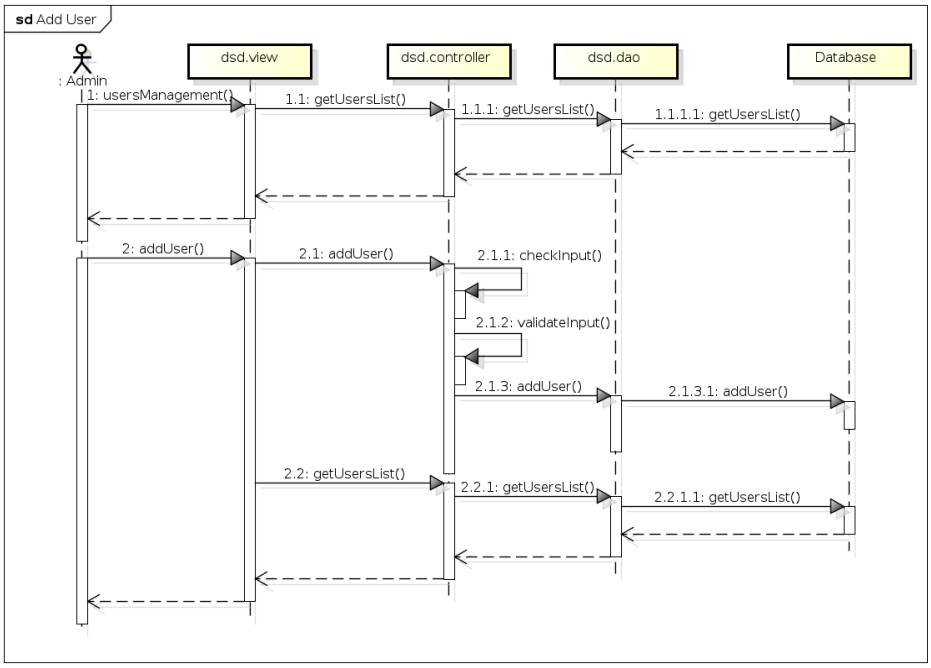
Regardless of this difficult constraints, we will deploy the data from the history to our server without accessing the ftp server and we will parse the data directly from our server.

Our server is a compact hardware unit consisting of 3 major components:

- MySQL database
- Apache tomcat web application server
- Input data local storage (storing the real input text files)

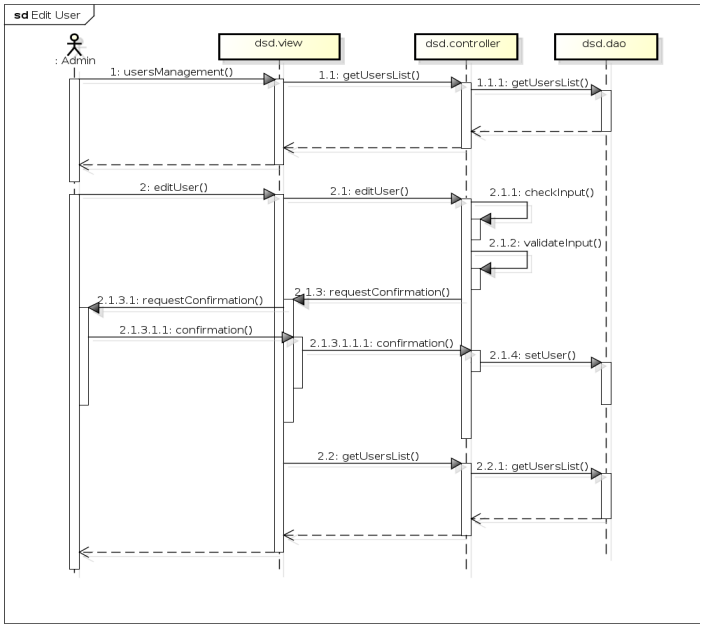
4.2 Data flow / Interactions / Dependencies

4.2.1 Add User



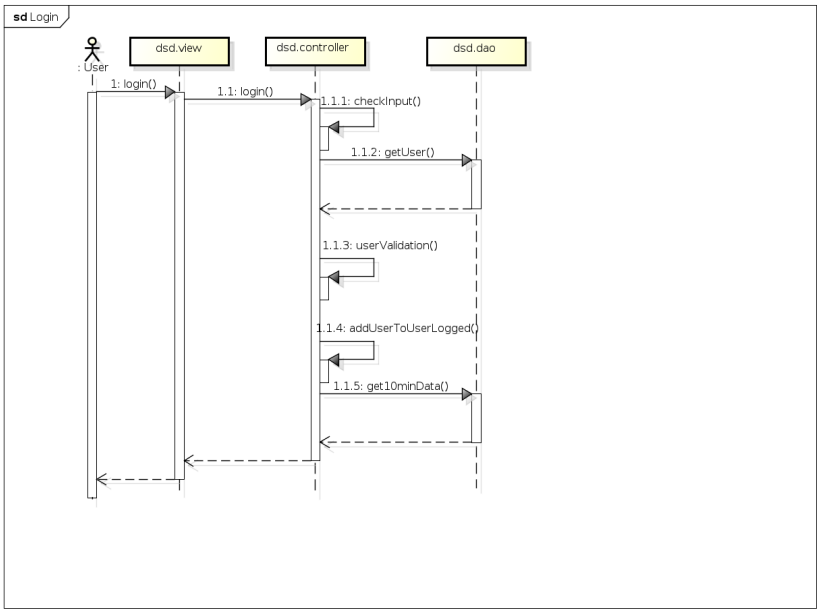
If the user wants to add a new user he go to the UserMangement. On this side he will provide Information above all user, fetching through the controlling. In this view, he can create a new users. This question will be provided by the view to the controller, who will perfom the task.

4.2.2 Edit User



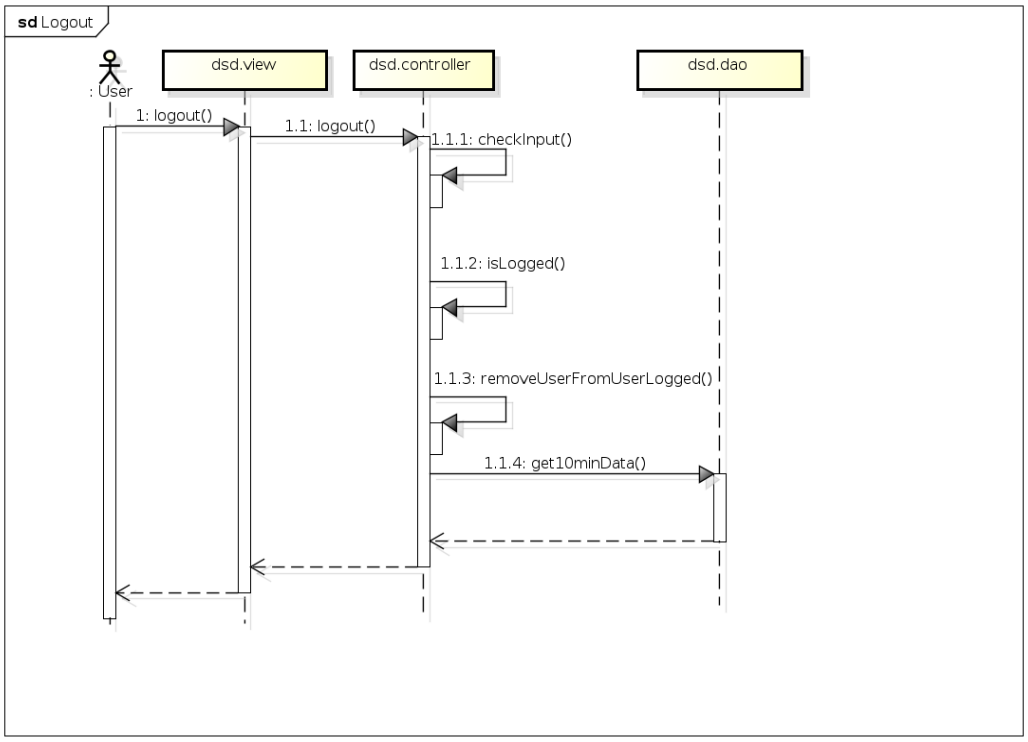
In the UserMangementView the administrator is able to edit a user. If he edit a user, the view give the call to the controller, who will perform the task.

4.2.3 Login



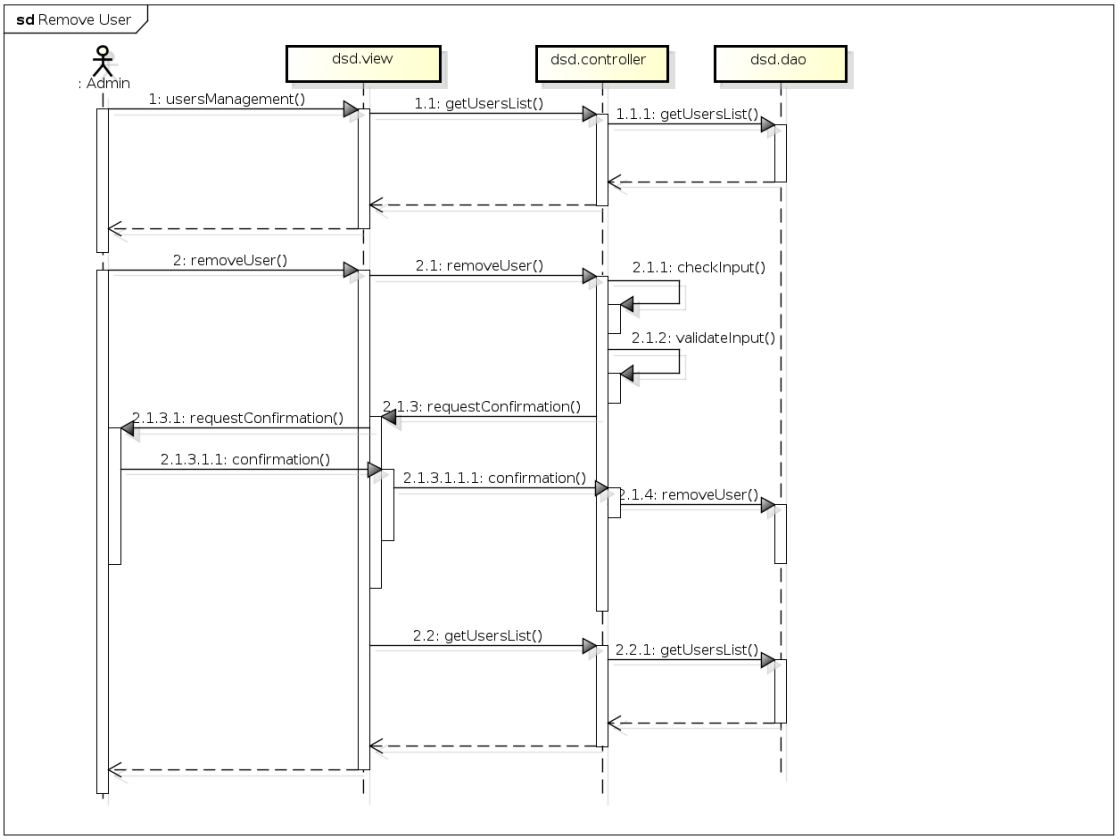
If a user logs in, the view will send the credentials to the controller, who will fetch the necessary data from the dao and check the login-request.

4.2.4 Logout



If the view gets a logout-request, he will provide the info to the controller, who performed the task.

4.2.5 Remove User



The Administrator can over the userMangementView remove a User. If the view regonize such a request, it will provide the information to the controller. The controller will perfome the taks.

4.3 Data Types / Formats

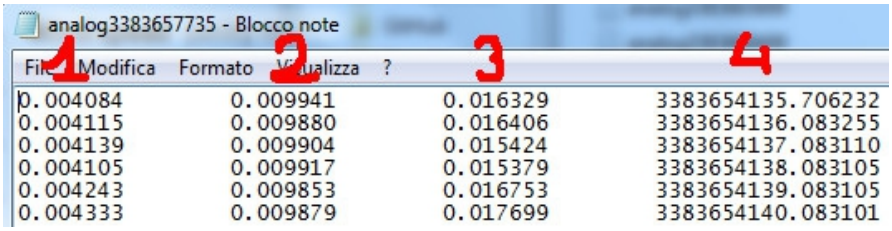
As data source for the Database adf.

There are three kind of inputs in two kind of formats:

- analog****.txt
- sonar*****.txt
- picture****.jpg

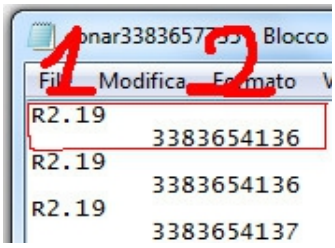
The analog file contains:

1. **Wind speed** (unity measure mA)
2. **Distance between the Hydrometer and the level of water** (unity measure mA)
3. **Wind direction** (unity measure mA)
4. **Timestamp of the detection of the sample** (Labview encode → see before) [*decimals can be dropped]*



The sonar file contains 2 columns of values, offset of a line (*fig.5*):

1. **Distance between the sonar and the bottom of the river** (unity measure meters)
2. **Timestamp of the detection of the sample** (Labview encode → see before)



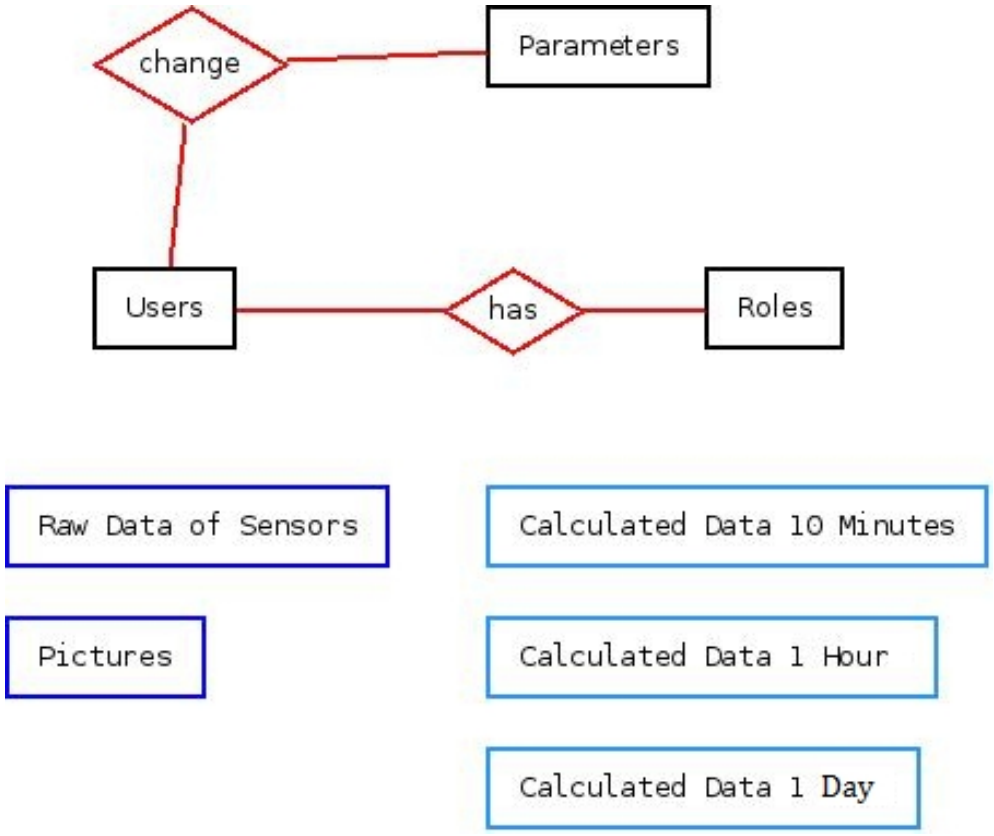
The images are two, one for each direction (Mantova – Modena):

(Mantova)

(Modena)

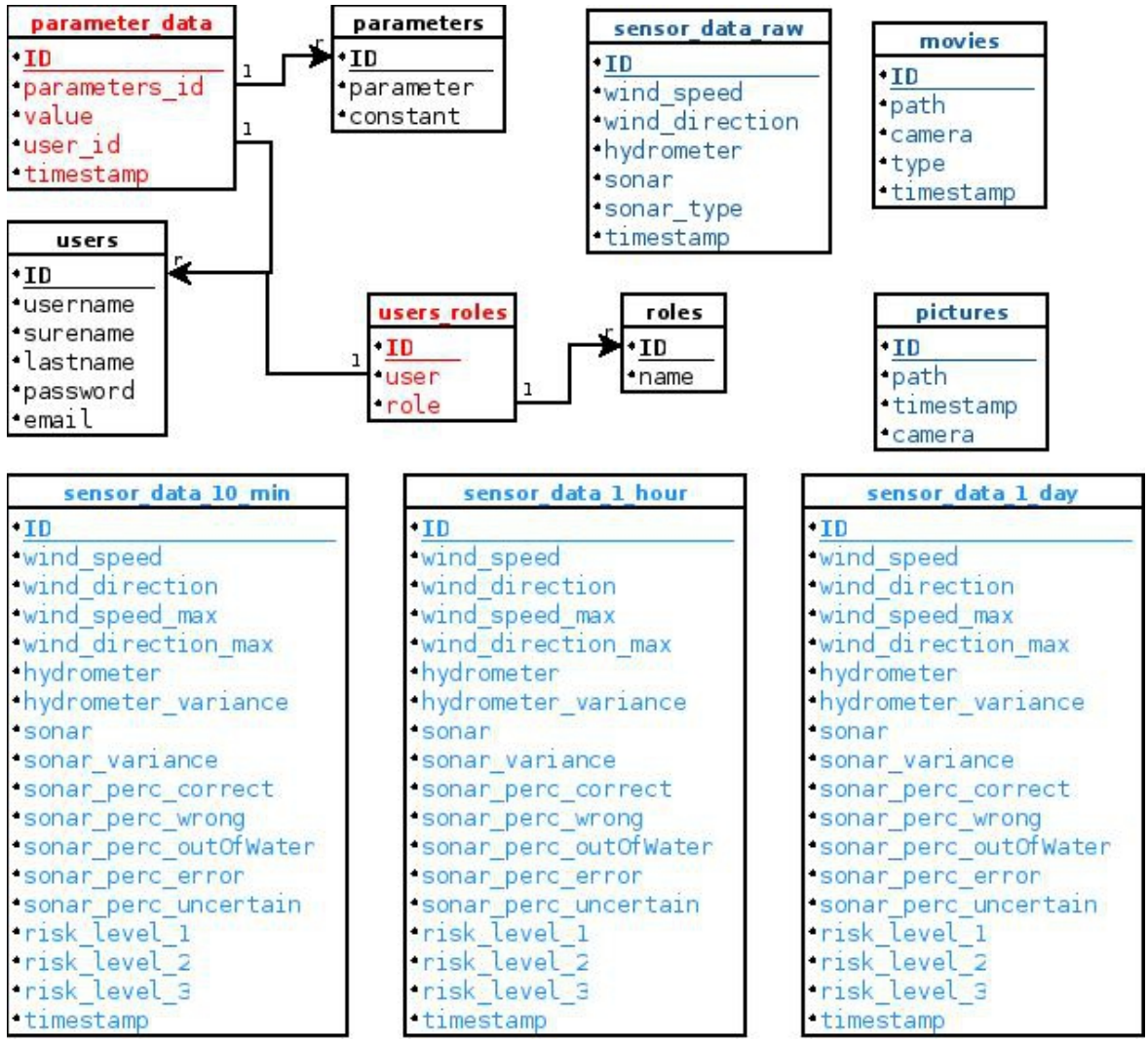
4.4 Database Model

In the Database we will two different kind of data. At the one side we have the sensor-data and calculated data and on the other side we will have administration data, like the users and their roles.



Entity-Relationship-Diagram of RTBM-Database

This Entity-Realtionship-Diagram shows an abstract view on the database. We will have four tables to store the raw and calculated data. Also we need one table to store meta-data like path and date of the pictures.



Detailed Diagram of RTBM-Database

For administration and the website-access we need to store the users and their role. Also we need to store the parameters and their values and changes.

4.4.1 Table structure for table parameters

The table “parameters” store all available parameters of the system and the information, if this parameter is changeable by the user.

Column	Type	Null	Default
ID	int	No	
parameter	varchar	No	
constant	tinyint	No	

4.4.2 Table structure for table parameter_data

The table “parameter_data” stores every change of an parameter and the according timestamp and the users, who changed the parameter. So this table also contains the actual value of every parameter.

Column	**Type**	**Null**	**Default**	**Links to**
ID	int	No		
parameters_id	float	No		parameters (ID)
value	int	No		
user_id	int	No		users (ID)
timestamp	timestamp	No	CURRENT_TIMESTAMP	

4.4.3 Table structure for table pictures

The table “picture” provide meta-information to every picture from the both webcames.

Column	**Type**	**Null**	**Default**
ID	int	No	
path	varchar	No	
timestamp	timestamp	No	CURRENT_TIMESTAMP
camera	int	No	

4.4.4 Table structure for table movies

The table “movies” provide meta-information to every movie created from the pictures.

Column	**Type**	**Null**	**Default**
ID	int	No	
path	varchar	No	
type			
timestamp	timestamp	No	CURRENT_TIMESTAMP
camera	int	No	

4.4.5 Table structure for table roles

The table “roles” holds the roles the user can have.

Column	**Type**	**Null**	**Default**
ID	int	No	
name	int	No	

4.4.6 Table structure for table sensor_data_1_day

In the table “sensor_data_1_day” the precalculated-mean-data for every day will be stored.

Column	**Type**	**Null**	**Default**
ID	int	No	
wind_speed	float	No	
wind_direction	float	No	
wind_speed_max	float	No	
wind_direction_max	float	No	
hydrometer	float	No	
hydrometer_variance	float	No	
sonar	float	No	
sonar_variance	float	No	
sonar_perc_correct	float	No	
sonar_perc_wrong	float	No	
sonar_perc_outOfWater	float	No	
sonar_perc_error	float	No	
sonar_perc_uncertain	float	No	
risk_level_1	float	No	
risk_level_2	float	No	
risk_level_3	float	No	
timestamp	timestamp	No	CURRENT_TIMESTAMP

4.4.7 Table structure for table sensor_data_1_hour

In the table “sensor_data_1_hour” the precalculated-mean-data for every hour will be stored.

Column	**Type**	**Null**	**Default**
ID	int	No	
wind_speed	float	No	
wind_direction	float	No	
wind_speed_max	float	No	
wind_direction_max	float	No	
hydrometer	float	No	
hydrometer_variance	float	No	
sonar	float	No	
sonar_variance	float	No	
sonar_perc_correct	float	No	
sonar_perc_wrong	float	No	
sonar_perc_outOfWater	float	No	
sonar_perc_error	float	No	
sonar_perc_uncertain	float	No	
risk_level_1	float	No	
risk_level_2	float	No	
risk_level_3	float	No	
timestamp	timestamp	No	CURRENT_TIMESTAMP

4.4.8 Table structure for table sensor_data_10_min

In the table “sensor_data_10_min” the precalculated-mean-data for every 10 minutes will be stored.

Column	Type	Null	Default
ID	i	No	
wind_speed	float	No	
wind_direction	float	No	
wind_speed_max	float	No	
wind_direction_max	float	No	
hydrometer	float	No	
hydrometer_variance	float	No	
sonar	float	No	
sonar_variance	float	No	
sonar_perc_correct	float	No	
sonar_perc_wrong	float	No	
sonar_perc_outOfWater	float	No	
sonar_perc_error	float	No	
sonar_perc_uncertain	float	No	
risk_level_1	float	No	
risk_level_2	float	No	
risk_level_3	float	No	
timestamp	timestamp	No	CURRENT_TIMESTAMP

4.4.9 Table structure for table sensor_data_raw

The table “sensor_data_raw” contains all raw data.

Column	Type	Null	Default
ID	int	No	
wind_speed	float	No	
wind_direction	float	No	
hydrometer	float	No	
sonar	float	No	
sonar_type	int	No	
timestamp	timestamp	No	CURRENT_TIMESTAMP

4.4.10 Table structure for table users

The table “users” contains all users and the according to information.

Column	Type	Null	Default
ID	int	No	
username	varchar	No	
surename	varchar	Yes	NULL
lastname	varchar	Yes	NULL
password	varchar	No	
email	varchar	Yes	NULL

4.4.11 Table structure for table users_roles

The table “users_roles” contain the relationship between users and roles.

Column	Type	Null	Default	Links to
ID	int	No		
user	int	No		users (ID)
role	int	No		roles (ID)

4.4.12 Used Libraries to Excess-Database

Since our program only communicate over Java with the MySQL server, we can use JDBC for every communication with the Database.

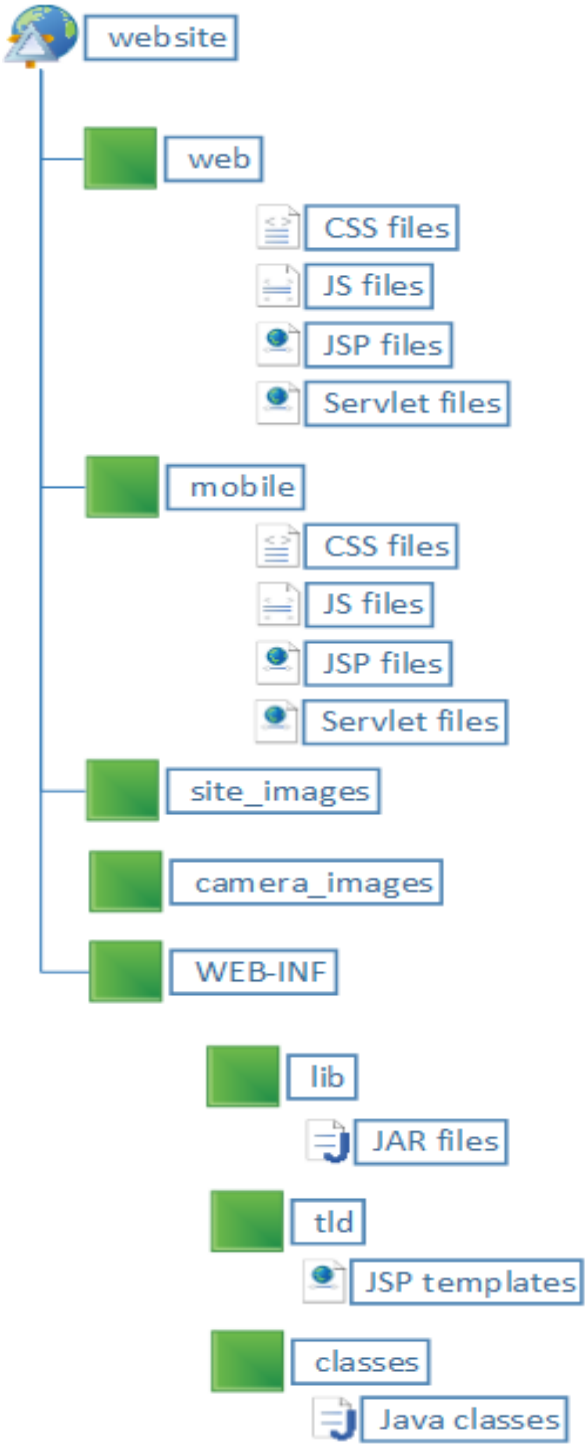
4.4.13 Security

For security reason we use a own user for access to database. This user has only access to the RTMS-DB and can only create local connections to database.

4.5 Web site organization

4.5.1 Web site organization

We are following the usual tomcat web application organization expanded with other components needed for our web application.



4.5.2 Website organization structure described

Website Folder	**Files included**
web	All the style sheet files, .jsp files, html files, javascript files, etc.
mobile	The web application of the project
site_images	Static images used for the page design
WEB-INF/lib	Other external java libraries needed for our project
WEB-INF/tld	Folder for storing the templates
WEB-INF/classes	Folder where our classes and our java code goes