# Technical Report
## Baltic Sea Weather Data

***Assignment:*** *Technical Report, Learning Task 5, Group 1, Theme 2.1-I*

***Authors:*** *Tom Broenink, Ward Holthof, Yuri Hoogeweg,*
*André Nanninga, and Maurits van Mastrigt*

***Date:*** *June 30th 2014*

# Table of Contents

# 1 Introduction

The United Nations Weather Data Management Institute (UNWDMI) collects weather data from all over the world. Because of the potential to attract customers from all over the globe, the Da Vinci Data division was set up to handle the European customers. The Aleksandro Stulginskio Universitetas (ASU) contacted the Da Vinci Data division to setup a meeting to discuss potential business.

They will perform calculations on weather data to conduct research on climate change. The university is interested in climate change in Lithuania and other countries around the Baltic Sea. The university is especially interested in weather changes regarding the milder weather in the past years (sleet, rainfall, umidity, and temperature).

This report describes the process of translation the customer wishes into a technical project specification for building the requested web application. Also some test results are included, to validate the speed requirements and to reinforce the propositions made in the Service Level Agreement.

# 2 Application Specification

The meeting, held on June 6th 2014 at the Hanzehogeschool Groningen, gave the Da Vinci Data division many insights on the requested application of the ASU. After the, rather pleasand meeting, the minutes were made and verified with the ASU. They agreed with the general outline of the application, which will be given in the next paragraph.

## 2.1 Requirements

The first meeting resulted in a general outline of the application, which in turn exists of the following requirements:

Aleksandro Stulginskio Universitetas will perform calculations on weather data to conduct research on climate change. The university is interested in climate change in Lithuania and other countries around the Baltic Sea. The university is especially interested in weather changes regarding the milder weather in the past years. They would like 'sleet' measurements.

Also they are interested in the following measurements: **Wind speed, rainfall, temperature, and humidity**.

In the following (relevant) countries:

- Lithuania
- Denmark
- Germany
- Poland
- Kalingrad (Russia)
- Estonia
- Latvia
- Finland
- Sweden

### Queries

These global functional requirements in turn translate into more detailed queries:

1. **Rainfall above 10 mm within 50 km.**
   Show data about rainfall when it is above 10 mm. Don't show when rainfall is below that. Only stations within 50 km of coastline of the Baltic Sea.
2. **Graphs of temperature and humidity.**
   Be able to show a graph of temperature and humidity of any individual weatherstation. (Since previous midnight). Any weatherstation worldwide.

## Key requirements

The following key requirements were agreed upon:

- Cockpit-like graphical presentation, no 'boring' tables;
- History of data by days, weeks and months;
- Half a year (6 months) data retention;
- Support Google Chrome and latest version of Internet Explorer;
- Data and webinteface will be hosted by Da Vinci Data;
- Map with colour code for wind speed and temperature values;
- Authenticate using username and password, no specific user roles;
- Allow for data to be downloaded in Excel file for the past half a year in days/weeks/months;
- ONLY measurements per minute;
- SSL to secure safe data transport.

## Load time for queries

Finally the maximum load time for the queries was discussed. The agreed time is 1 minute for both queries.

# 2.2 Goals

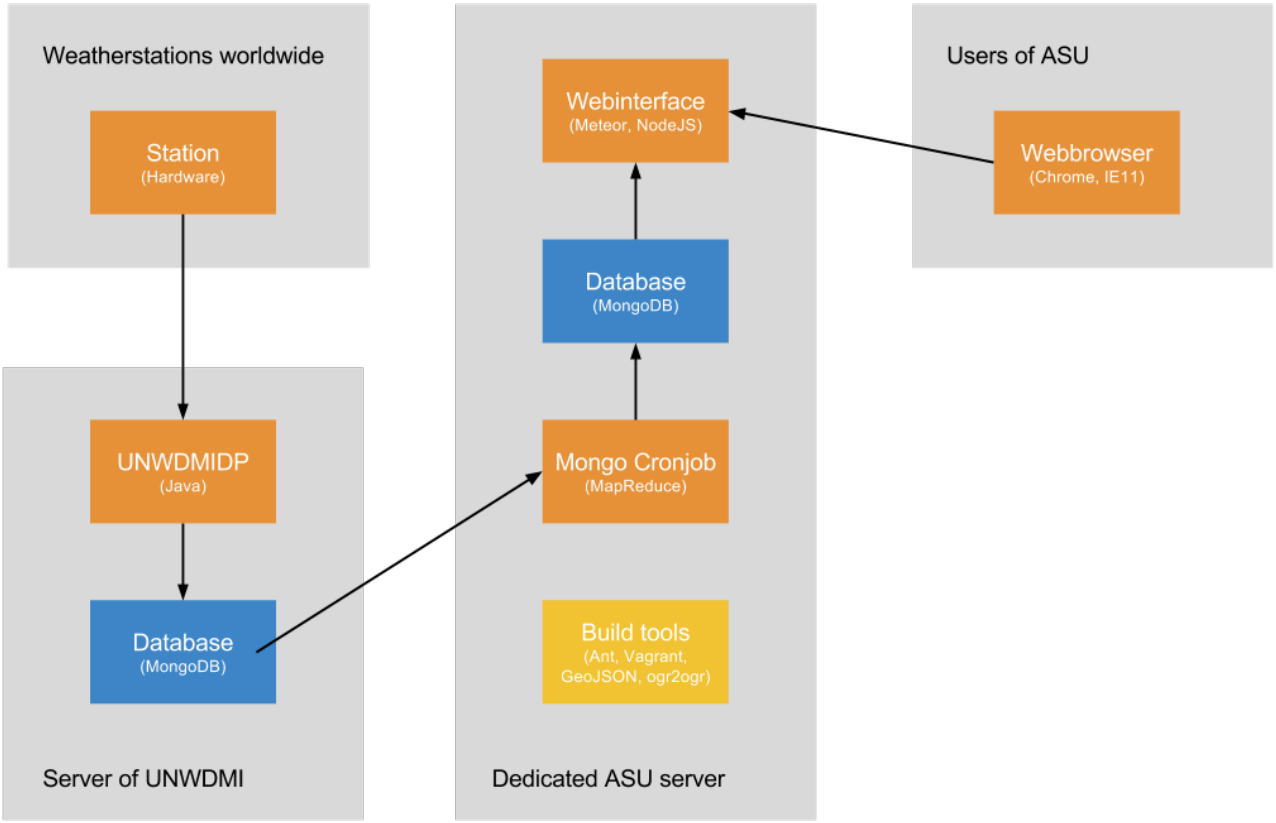
The goals that can be derived from the requirements are:

1. Cockpit-like and responsive graphical webinterface;
2. World map with stations linked to graphs of temperature and humidity;
3. Baltic Sea map with stations, within 50km around the coast, linked to a rainfall graph;
4. Measurements per minute;
5. Half a year data retention, with daily backups;
6. User authentication and secure transfer of the data.

# 3 Applicaton Design

The following chapters will each describe a crucial part of the design to accomplish the customers' requirements.

This infographic will give a better overview of the complete infrastructure and dataflow:




## 3.1 UNWDMIDP

The United Nation Weather Data Management Institute Data Processor (UNWDMIDP) is the application used to gather the weatherdata as measured by the 8000 weatherstations available to Da Vinci Data. The Data Processor collects the measurements of each weatherstation every second. It checks if any data is missing or malformed and repairs those measurements by looking at the trend of previous measurements. It stores the raw measurements in a Mongo database which is then reduced to the actual working dataset as detailed in the Map Reduce section of this report.

The Data Processor was developed to be as lightweight as possible as to handle the 8000 measurements per second.

## 3.2 Ant

To simplify the development of the application the build tool Ant was used. Ant is a Java built tool by Apache which is used to automate common tasks. These tasks could then be easily execute as to save time on development. By executing for example `ant setup` multiple tasks are executed to setup the entire development environment. Within the setup tasks the database is setup to create the appropiate collections and indexes, and import the weatherstations from a JSON file into the database. And the vector maps as detailed in the GeoData section are converted to GeoJSON to be used within the application.

Automated these tasks means that they are easier to use and any possible human errors are greatly reduced.

## 3.3 Vagrant

Vagrant is a tool to create lightweight virtual development environments. It is used for the development of the application to ensure that the developers would not run in platform specific issues. The virtual machine hosted a Ubuntu installation installed with all the tools and software needed to run the application. Using the command `vagrant up` the machine is easily started and setup for development.

Vagrant proofed to be very helpful by ensuring that every developer could work on the application regardless of operation system.

## 3.4 Meteor

As the basis of the application Meteor was used. Meteor is JavaScript framework for both clientside as serverside. It features a terrific data synchronisation library which greatly helps with delivering realtime and up-to-date data. Meteor is built in such a way that whenever a change is made to the database (the Mongo database) those changes are automatically and instantly synchronized with the client. This means that no extra code had to be written to create a realtime view of all the measurements.

Meteor provides a rich set of packages which are thoroughly tested and widely supported. One such packages that was implemented is the 'accounts-password' package. This prebuilt package provides all the functionality for accounts that was required. Other packages which we could easily and directly use include: D3, Jquery, Iron-Router, Underscore and Bootstrap.

## 3.5 Map Reduce

The requirements as set by the ASU were that the measurements should be provide with a frequency of 1 measurement per minute. The actual measurements are however made with a frequency of 1 measurement per second. To meet the requirement a Map Reduce was implemented that collects every measurement of a weatherstation, calculats the average of the measurements and finally saves these averages in a new collection. This Map Reduce is executed every minute on the previous minute of measurements.

Finally the collection containing the averages is used within the application. All the data in both the map and the line graphs is provided by this collection of the averages per minute.

## 3.5 D3

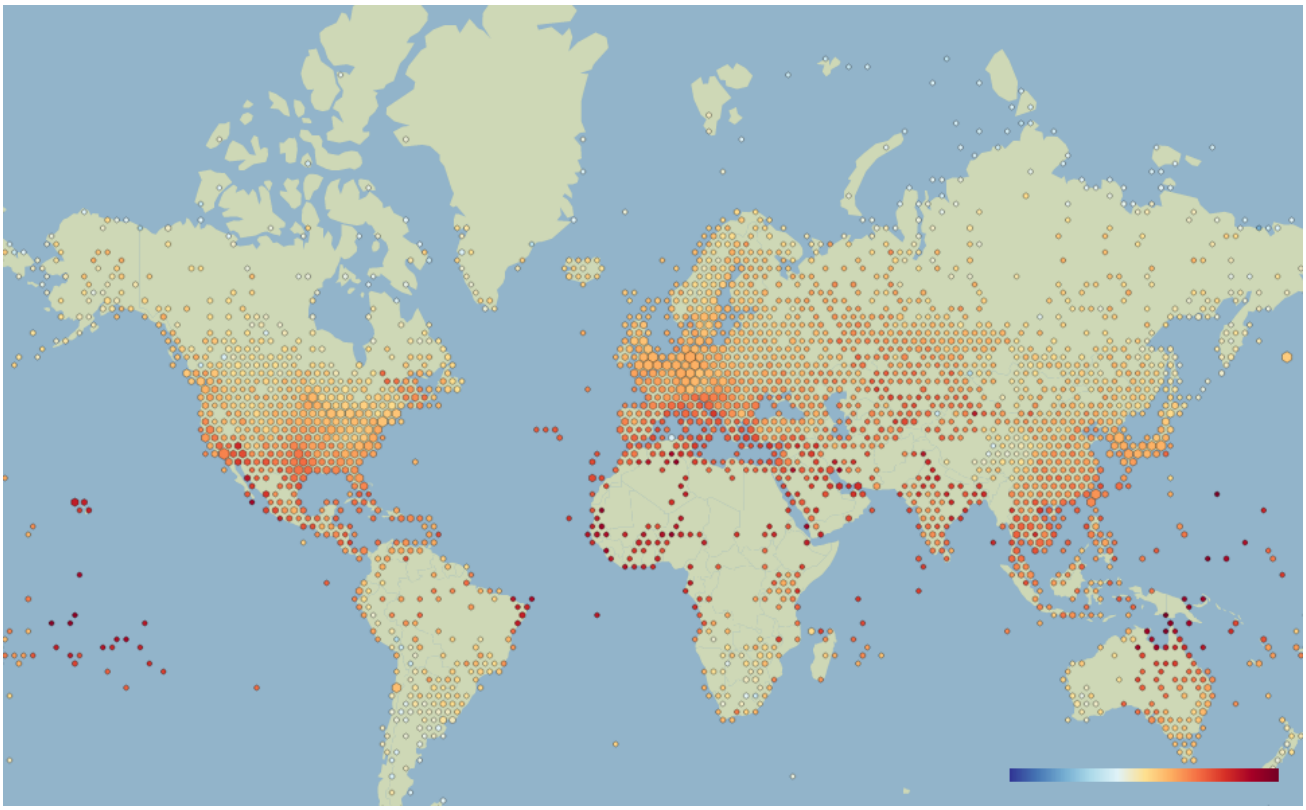
D3 is a JavaScript library for manipulating document based on data. With this library the weatherdata is visualised in such a way that it is easily digestable. The D3 library is used for both the map, giving a geographical impression of the data, and the linegraphs, showing detailed historical values for temperature, precipitation and humidity.

## Line Graph

Line graphs are used to give a detailed view of the measurments of a weatherstation. The line graphs are built using the D3 library and contain data for each weatherstation within a hexagon. The graphs details the measured data over a period of a day, week or month. This can give a good insight in the trends of the measured data and the relation between weatherstations within a hexagon.

# Geodata

To visual the data in a clear and concise manner a map with the weatherstations was used. This map gives both insight into the location and relation between weatherstations and into patterns that the weather may have in specific geographical areas. The map of the wordt with the hexagons showing the temperature gives a very good overview.



*World map*

The figure above gives a very clear picture of the percipitation around the baltic sea. Every hexagon on this map contains one or more weatherstations. The size of the hexagon gives an indication on the amount of weatherstations in that hexagon while the color gives an indication on the amount of percipitation as measured by those weatherstations.

# Natural Earth Data

The maps are made using vector maps provided by www.naturalearthdata.com. These are converted to the GeoJSON format which can be used by D3.

To convert the maps both to GeoJSON both ogr2ogr and TopoJSON are used. Ogr2ogr is used to simplify and reduce the maps to only the area we need and TopoJSON then converts that to the GeoJSON format.

The code below shows how these tools were used.

```
ogr2ogr \
 -simplify .001 \
 -f GeoJSON \
 -where "ADM0_A3_IS IN ('SWE', 'DNK', 'DEU', 'POL', 'RUS', 'LTU', 'LVA', 'EST', 'F
 /vagrant/temp/countries.json \
 /vagrant/resources/ne_50m_admin_0_countries/ne_50m_admin_0_countries.shp
topojson \
 -o /vagrant/src/public/geo/baltic.json \
 -p name \
 - /vagrant/temp/topo_countries.json
```

# Hexbinning

To show the measurement a method called Hexbinning was used. This is a method to simplify many data points on a map to give a better visual indication of the actual measurements. With this methods data point are grouped in hexagons which can then be styled to give an indication fo the data within.

In the application the hexagons are styled in a way to indicate two possible values. Firstly the size of the hexagons give an indication of the amount of weatherstations within a hexagon. Secondly the color of the hexagon is determined by the average measurement of the weatherstations within a hexagon. For the baltic map the color indicates the amount of percipitation above 10mm, the hexagon is colored grey when below 10mm. For the world map the temperature is visualised by coloring the hexagons.

# 3.7 Backups

Backups are essential in case of hardware failure and data loss. As described in the Service Level agreement backups will be made on a daily basis. Because the dataset itself is large and will only grow, making a full backup isn't an option. Unfortunately with a NoSQL database like MongoDB incremental backups are also not a built in feature and must be done manually.

Luckily with MongoDB there is a far more clever approach: Replication with slave delay. This essentially is a full backed up running copy of the full server, but with synchronization delay of a half day (a reasonable restoring time). This ensures always having the data of the previous the last day.

Setting up delayed master-slave replication with MongoDB is relatively easy. It can be done by running the following command on a seperate server:

```
mongod --port 37017 --slave --slavedelay 43200 --source <master server>:27017 --db
```

The slave delay is specified in seconds (43200 = 12 hours) and all data is stored in the `/tmp/backup` folder.

Each weather measurement that this assignment requires takes up *240 bytes* of diskspace. Which means each day takes only `60 * 24 * 240 ~= 0.33 MB` diskspace. This is *59,68 MB* worth of data for half a year.

A downside -that should be mentioned- is the requirement of having a server with the same disk capacity as the original server. Moving old data from the slave server to a backup disk can be done, but has to be investigated more thoroughly. This downside is currently not relevant, since the assignment requirements only require a half a year data retention.

More information about master-slave replication can be found on the MongoDB website.

# 4 Test Results

To determine the workload the application is capable of handling, some tests have done. First of the processing of weather data will be tested. Then the first and the second query (as described in chapter 2) will be tested. Finally, the drawn conclusions will be given, which are based on the test results.

## 4.1 Weather Data

In the previous assignment (learning task 2) the throughput of the weather data processor was tested. Here are the results:

| Duration | Clusters | Memory | Mutations | # of records | Expected # of records | Efficiency |
|---|---|---|---|---|---|---|
| 15 minutes | 800 | 153.50 MB | 7207550 | 7206750 | 7200000 | 100.09% |
| 30 minutes | 800 | 149.00 MB | 14176000 | 14472520 | 14400000 | 100.05% |
| 60 minutes | 800 | 148.50 MB | 28805760 | 28804960 | 28800000 | 100.02% |

After executing numerous long term running stresstests, the application was still able to process without data loss. The efficiency remains 100% en the amount of used memory is stable. After a runtime of 15 minutes the size of the database was *1.73 GB*, and after 1 hour runtime it was *6.91 GB*. Which is almost exactly 4 times larger. Also a runtime of 30 minutes results in a database with a size of *3.46 GB* (twice the size of a 15 minute runtime). From this can be concluded that the database has a lineair growth, with a size increase of approximately *6.9 GB* per hour.

# 4.2 First Query

The first query is **rainfall above 10 mm within 50 km of the Baltic Sea coast**.

The response should be served within 1 minute. Because there are many things to take in consideration, a simple timing test seems te most appropriate. During a period of 20 minutes the Baltic Sea page was fully refreshed 10 times. This page contains a map with 188 weatherstations and a rainfall graph per station. This test resulted in the following load times:

| Test | Start time | Stop time | Response time |
|---|---|---|---|
| 1 | 11:16:01 | 11:16:04 | 3.2 seconds |
| 2 | 11:16:20 | 11:16:25 | 4.7 seconds |
| 3 | 11:16:28 | 11:16:32 | 3.5 seconds |
| 4 | 11:22:36 | 11:22:40 | 3.8 seconds |
| 5 | 11:22:54 | 11:23:04 | 8.4 seconds |
| 6 | 11:23:10 | 11:23:15 | 4.2 seconds |
| 7 | 11:23:18 | 11:23:22 | 3.4 seconds |
| 8 | 11:30:20 | 11:30:04 | 3.8 seconds |
| 9 | 11:30:28 | 11:30:32 | 4.5 seconds |
| 10 | 11:30:40 | 11:30:44 | 3.7 seconds |
| **Average** | | | 4.3 seconds |

From this can be concluded that the response times are well within the 60 second limit.

Because data is stored in a temporary and local Mongo database, a big part of the load is made redundant, since loading the stations and old weather data will be cached. Therefore the number data requests will be limited, which will lower the server and database load.

# 4.3 Second Query

The second query is **graphs of temperatue and humidty for any weatherstation worldwide**.

This response should also be served within 1 minute. For the second query there are the same things to take in consideration as the first query, except for the larger amount of weather stations (8000 instead of 188) and removal of the limitation of not showing stations (with rainfall less than 10 mm). Also a simple timing test seems te most appropriate in this case. During a period of 20 minutes the World map page, with map and graphs, was fully refreshed 10 times. This resulted in the following load times:

| Test | Start time | Stop time | Response time |
|---|---|---|---|
| 1 | 11:32:22 | 11:32:38 | 16.3 seconds |
| 2 | 11:32:42 | 11:33:00 | 17.2 seconds |
| 3 | 11:33:05 | 11:33:22 | 16.6 seconds |
| 4 | 11:40:48 | 11:41:14 | 25.8 seconds |
| 5 | 11:41:16 | 11:41:32 | 16.2 seconds |
| 6 | 11:41:37 | 11:41:52 | 16.3 seconds |
| 7 | 11:50:06 | 11:50:23 | 16.8 seconds |
| 8 | 11:50:29 | 11:50:49 | 20.2 seconds |
| 9 | 11:50:54 | 11:51:11 | 16.6 seconds |
| 10 | 11:51:17 | 11:50:34 | 16.7 seconds |
| **Average** | | | 17.9 seconds |

From this can be concluded that the response times are well within the 60 second limit.

The world map uses the same functionality as the Baltic Sea map (with a few differences). Because of this the data is also stored in a temporary and local Mongo database, which in turn also lowers the number of data requests and server and database load.

# 4.3 Conclusion

The first query has an average load time of *4,3 seconds*, which allows for a network delay of appromixately *55 seconds*. The second query has an average load time of *17,9 seconds*, which allows for a network delay of approximately *42 seconds*. Both are well within the measured network delay between the Netherlands and Lithuania:

```
~ $ ping -c 5 www.teo.lt
PING www.teo.lt (212.59.0.11): 56 data bytes
64 bytes from 212.59.0.11: icmp_seq=0 ttl=56 time=59.905 ms
64 bytes from 212.59.0.11: icmp_seq=1 ttl=56 time=60.553 ms
64 bytes from 212.59.0.11: icmp_seq=2 ttl=56 time=60.519 ms
64 bytes from 212.59.0.11: icmp_seq=3 ttl=56 time=60.428 ms
64 bytes from 212.59.0.11: icmp_seq=4 ttl=56 time=59.818 ms

--- www.teo.lt ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 59.818/60.245/60.553/0.317 ms
```

*Ping to Lithuania*

Also the databases uses a B-tree index which scales *O(log n)*. Since this will be a small part of the load time -because of rendering, network delay, and logic-, it is safe to say the queries will not reach the 60 second limit any time soon.

# 5 Glossary

This is an alphabetical list of technical terms used in this SLA with their definitions.

- **Ant**

> *A build tool for automating tasks and processes.*
> *— https://ant.apache.org/*

- **Database**

> *A database is a data structure that stores organized information.*
> *— http://www.techterms.com/definition/database*

- **D3**

> *A JavaScript library for manipulating documents based on data.*
> *— http://d3js.org/*

- **GeoJSON**

> *A format for encoding a variety of geographic data structures.*
> *— http://json.org/*

- **Hexbinning**

> *A method to reduce data points in to a visually clear representation*
> *— http://geospatial.commons.gc.cuny.edu/2013/11/25/hexbinning/*

- **JSON**

> *JavaScript Object Notation, a lightweight data-interchange format.*
> *— http://json.org/*

- **Map Reduce**

> *A data processing paradigm for condensing large volumes of data into useful aggregated results.*
> *— http://docs.mongodb.org/manual/core/map-reduce/*

- **Meteor**

> *An open-source platform for building web applications.*
> *— https://www.meteor.com/*

- **ogr2ogr**

  > *A tool to convert Vector Maps into other data formats.*
  > *— http://www.gdal.org/ogr2ogr.html*

- **Server**

  > *A server is a computer that provides data to other computers. It may serve data to systems on a local area network (LAN) or a wide area network (WAN) over the Internet.*
  > *— http://www.techterms.com/definition/server*

- **SSL Connection**

  > *Stands for "Secure Sockets Layer." SSL is a secure protocol developed for sending information securely over the Internet. Many websites use SSL for secure areas of their sites, such as user account pages and online checkout. Usually, when you are asked to "log in" on a website, the resulting page is secured by SSL.*
  > *— http://www.techterms.com/definition/ssl*

- **TopoJSON**

  > *A tool to convert Vector Maps into GeoJSON format.*
  > *— https://github.com/mbostock/topojson*

- **Vagrant**

  > *Software for creating and configuring virtual development environments.*
  > *— https://www.vagrantup.com/*

- **Vector Map**

  > *A vector based collection of Geographic Information System data.*
  > *— http://en.wikipedia.org/wiki/Vector_map*