

PROJECT DOCUMENTATION

ITV2E-1
RANDOMIT

Project

2.2

Auteur:	Teun de Jong	308158
	Stefan Jilderda	406347
	Jens Maas	349557
	Stefan Kuppen	405611
	Maurice Pater	397390
	Romano Braxhoofden	405380
Klas:	ITV2E-1	
Datum:	01-02-2021	
Plaats:	Groningen	
Opleiding:	HBO-ICT	
Onderwijsinstelling:	Hanzehogeschool	

Contents

Project documentation.....	1
1. Use case.....	1
2. Achieved goals of the application:	1
3. Specifications (Client requirements)	1
3.1 Application.....	1
4. Design.....	2
5. Java application	3
5.1 Rundown / general explanation	3
5.2 Synchronization	6
6 Server configuration	7
6.1 Installation, info, and elaboration on choices	7
6.2 Throughput of data	8
6.3 Transport of data.....	8
6.4 Files / folders structure	9
7 Data visualization (front-end).....	10
7.1 Login	10
7.2 Map.....	10
7.3 Station information table	11
7.4 Downloading data	12
7.5 Change password	13
8 Stress testing	14

Project documentation

1. Use case

The use case for the program is to receive, filter and write useable data from the weather station generator provided to us. The data gets stored on a server and is available through the provided web portal. Here you can search and view current, latest and previous data of specific weather stations. Through the web portal you can also view data up to 7 days ago, provided any data is known during these days.

2. Achieved goals of the application:

The application handles up to 800 incoming connections, with each connection sending data values of 10 different weather stations. These values get broken down to usable data which is then processed and filtered. After this process, the data is written to a specific location. Which in our case is a storage server connected through via a network bridge. Our application allows us to process weather data from up to 8000 weather stations. All data gets validated, filtered and stored.

A highly specific web portal has been created to view the stored data effectively, viewing up to 7 days of data as requested by the client.

The website contains the following features:

- Homepage accessible by everyone.
- A map accessible by logged in users.
- A login page.
- A password change page.
- Individual pages of every station depicting their information in tables.
- A top 5 of highest peak temperatures.

3. Specifications (Client requirements)

The following guidelines and requirements are met:

3.1 Application

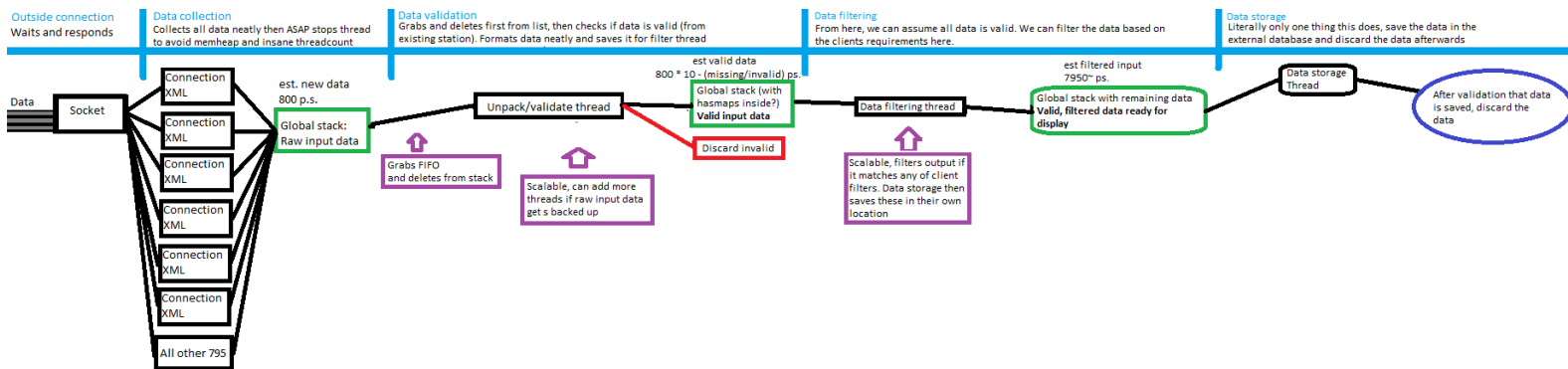
- The following data will be collected:
 - All weather station data around and inside the Pacific Ocean.
 - All weather station data around and inside of Iran but following the following requirement.
 - Temperature is equal or between 0 and 10 degrees Celsius.
- All data not matching the clients' needs will not be available for the client but will be stored.

3.2 Visualization

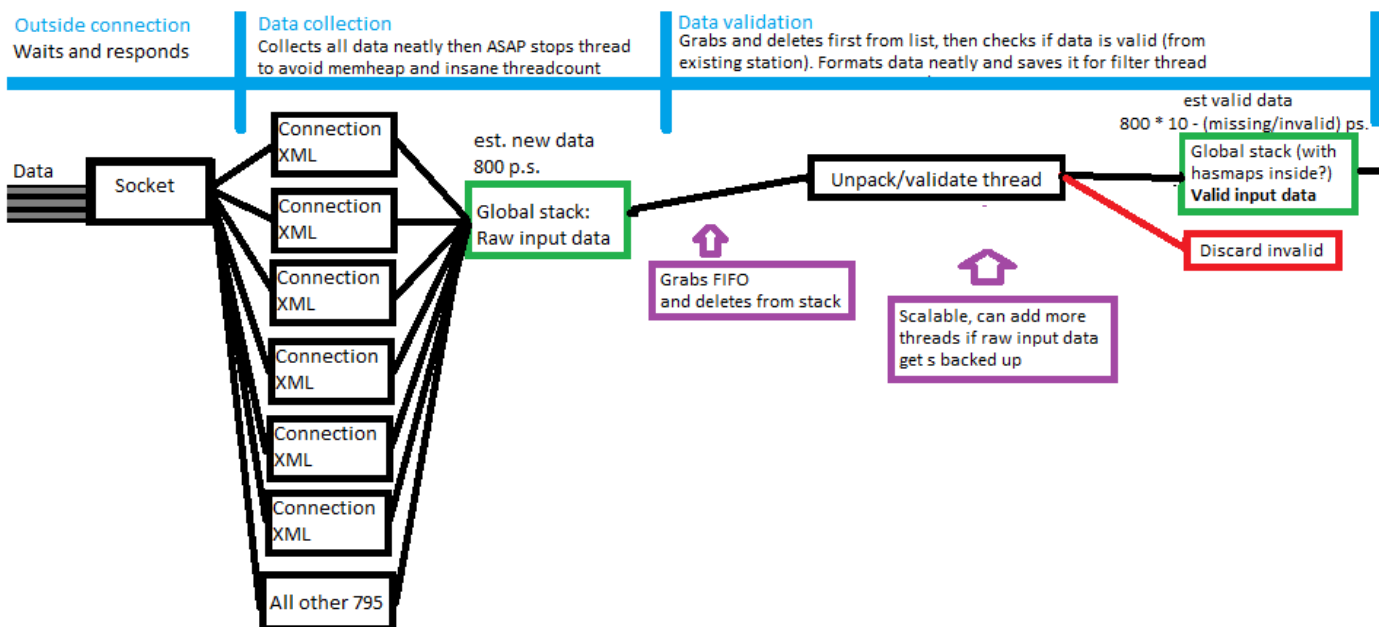
- Data will be displayed on a web-portal.
- The web portal is secured by HTTPS encryption.
- The web portal is secured by a user login system.
- The login system and accounts management are managed by RandomIT.
- The collected data is displayed on an interactive map.
- The data is sorted and filtered by station.
- Specific data can be searched for.
- A peak of temperatures is calculated and displayed.
- Missing data is re-calculated using extrapolation, based on the previous 30 data-points.

4. Design

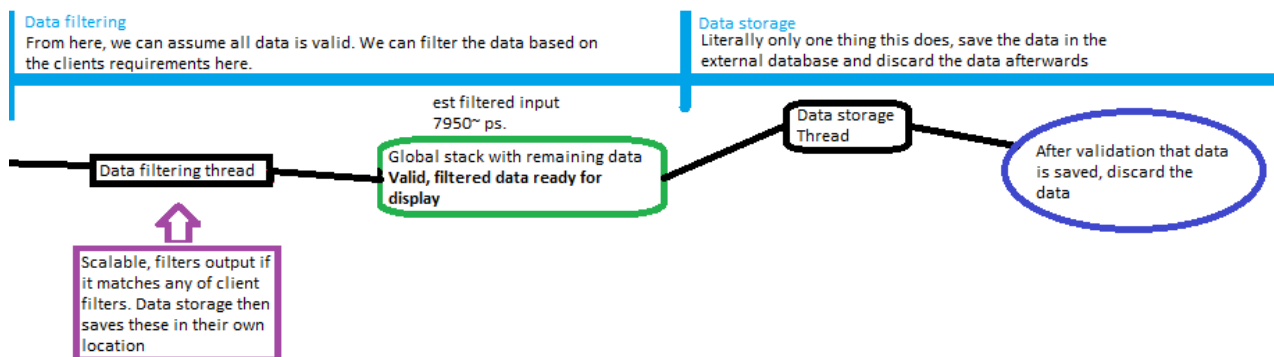
The design of the application has been carefully thought out to keep scalability in mind. The following figure illustrates the dataflow and logic behind the application:



Outside connection / Data collection / Data validation



Data filtering / Data storage



5. Java application

5.1 Rundown / general explanation

The application is cut in 5 parts which all make or generate their own synchronized threads to work:

5.1.1 Run

- Makes volatile queues for the threads to put output into (and grab from).
- Makes all starting threads and monitors queue sizes.

```
public class Run {
    public static volatile Queue<String> rawinput = new LinkedList<String>();
    public static volatile Queue<HashMap<String, String>> validinput = new LinkedList<HashMap<String, String>>();
    public static volatile Queue<HashMap<String, String>> filteredinput = new LinkedList<HashMap<String, String>>();
    private static final int port = 7789;

    Run | Debug
    public static void main(String[] args) {
        //Data receiver thread
        System.out.println("[LOG] Starting receiver thread");
        DataReceiver datareceiver = new DataReceiver("Receiver 1", port);

        //Data parser thread
        System.out.println("[LOG] Starting parser thread");
        Parser parser = new Parser("Parser 1");

        //Data filter thread
        System.out.println("[LOG] Starting filter thread");
        Filter filter = new Filter("Filter 1");

        //Data storage thread
        System.out.println("[LOG] Starting datastorage thread");
        DataStorage datastorage1 = new DataStorage("Datastorage 1");

        System.out.println("[LOG] Starting server thread");

        //Starting...
        datareceiver.start();
        parser.start();
        filter.start();
        datastorage1.start();
    }
}
```

5.1.2 Data Receiver

- Starts a main thread and opens a socket on port 7789.
 - Each connection that tries to connect gets its own thread, defined in Client-Thread.
 - Data gets sent and collected. Then gets stored in the raw-input queue.

```
public synchronized void run() {
    try (BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))){
        String out = "";
        String line = "";
        while (true) {
            line = in.readLine();
            out += (line + "\n");
            if (line.equals("</WEATHERDATA>")) {
                // Make sure the queue is not backed up. Otherwise ignore the requests
                if (Run.rawinput.size() < 16000) {
                    Run.rawinput.add(out);
                    out = "";
                }
            }
        }
    } catch (Exception e) {
        //e.printStackTrace();
    }
}
```

5.1.3 Parser

- Grabs latest from raw-input queue and checks if the data is valid. (also fills up any missing data).
 - Cross-references ID on a complete list of all valid ID's
 - Grabs all data from the XML.
 - Fills out missing data with "MISSING" which can be filled in later (frontend).
 - Parser outputs in the second volatile queue, valid input.

```
//Load all valid stations
ArrayList<String> validStationNumbers = new ArrayList<String>();
try (Scanner scanner = new Scanner(new File("./station_country_data.dat"))) {
    while(scanner.hasNextLine()) {
        validStationNumbers.add(scanner.nextLine().replaceAll("[^0-9]", "")); //only get the numbers of this file
    }
} catch (Exception e) {}
e.printStackTrace();

//Infinite loop to keep the thread alive
while (true) {
    if (Run.rawinput.size() > 0) {
        HashMap<String, String> returnHash = new HashMap<String, String>();
        try {
            String xmlString = Run.rawinput.poll();
            if (xmlString.contains("<MEASUREMENT>")) {
                for (String subString : xmlString.split("<MEASUREMENT>")) {
                    if (subString.contains("STN")) {
                        String stationNumber = subString.substring(subString.indexOf("<STN>") + 5, subString.indexOf("</STN>"));
                        if (stationNumber != null && !stationNumber.isEmpty() && validStationNumbers.contains(stationNumber)) {
                            String date = subString.substring(subString.indexOf("<DATE>") + 6, subString.indexOf("</DATE>"));
                            String time = subString.substring(subString.indexOf("<TIME>") + 6, subString.indexOf("</TIME>"));
                            String temperature = subString.substring(subString.indexOf("<TEMP>") + 6, subString.indexOf("</TEMP>"));
                            String dewPoint = subString.substring(subString.indexOf("<DEWP>") + 6, subString.indexOf("</DEWP>"));
                            String stationLevelPressure = subString.substring(subString.indexOf("<STP>") + 5, subString.indexOf("</STP>"));
                            String seaLevelPressure = subString.substring(subString.indexOf("<SLP>") + 5, subString.indexOf("</SLP>"));
                            String visibility = subString.substring(subString.indexOf("<VISIB>") + 7, subString.indexOf("</VISIB>"));
                            String windSpeed = subString.substring(subString.indexOf("<WDSP>") + 6, subString.indexOf("</WDSP>"));
                            String precipitation = subString.substring(subString.indexOf("<PRCP>") + 6, subString.indexOf("</PRCP>"));
                            String snowDrop = subString.substring(subString.indexOf("<SNDP>") + 6, subString.indexOf("</SNDP>"));
                            String events = subString.substring(subString.indexOf("<FRSHIT>") + 8, subString.indexOf("</FRSHIT>"));
                            String cloudCoverage = subString.substring(subString.indexOf("<CLDC>") + 6, subString.indexOf("</CLDC>"));
                            String windDirection = subString.substring(subString.indexOf("<WNDDIR>") + 8, subString.indexOf("</WNDDIR>"));

                            returnHash.put("StationNumber", stationNumber);
                        }
                    }
                }
            }
        } catch (Exception e) {}
    }
}
```

5.1.4 Filter

- Grabs latest input from valid-input and check if a client wants to have this info. Then adds a tag to the data. This can be done for multiple clients.
- The filter outputs filtered input in the latest volatile queue.

```
//Filter for Pacific ocean stations
try {
    if (pacificStationListClient1.contains(hashmap.get("StationNumber"))) {
        String temperature = hashmap.get("Temperature");

        if (!temperature.equals("MISSING")) {
            Double temp = Double.parseDouble(temperature);

            if (temp >= 0 || temp <= 10) {
                //Filtered data required by client comes through. Add client name to list
                clientlist.add("UniversityTeheran");
            }
        }
    }
} catch (Exception e) {
    //e.printStackTrace();
}
```

5.1.5 Data storage

- Data storage grabs latest data from filtered input, then stores the data in the global folder.
- Client data also gets stored in their own folder.

```
// Save ALL client data, always. No exceptions!
try {
    if (hashMap.get("Client").equals("")) {
        //No extra data saving is required
    } else { //Save the json in the client-folder aswell
        data_forclient = data_forclient + 1;

        //Remove [] and all special chars from string
        String rawclients = hashMap.get("Client").replaceAll("[^a-zA-Z, ]", "");

        //Split into array from different clients
        String[] clients = rawclients.split(", ");

        //Walk through the array
        for (String clientname : clients) {
            //Make sure the directory is created
            File directory1 = new File(defaultstoragelocation + clientname);
            if (!directory1.exists()) {
                directory1.mkdir();
            }

            try {
                //Constructs a FileWriter given a file name, using the platform's default charset
                file = new FileWriter(defaultstoragelocation + clientname + "/" + stationID + "-" + timestamp + ".json");
                file.write(json.toJSONString());
                data_succeeded = data_succeeded + 1;
            } catch (IOException e) {
                //e.printStackTrace();
            } finally {
                try {
                    file.flush();
                    //file.close();
                } catch (Exception e) {
                    //e.printStackTrace();
                }
            }
        }
    }
} catch (Exception e) {
    //e.printStackTrace();
}
```


5.2 Synchronization

All threads used are synchronized. This means they can all access the latest data in the 3 volatile queues. All threads do their own job or sleep for 1 second, then try again. This way all threads know about the queues and how full they are. If there is work to be done, they will start doing it immediately.

5.2.1 Queues

Our Java application makes active use of 3 major queues where all data is stored if the data cannot move on to the next step in our process. Making these queues volatile means they will always be correct (and loaded into memory) when a thread tries to access the information (for example, count or poll).

```
public class Run {  
    public static volatile Queue<String> rawinput = new LinkedList<String>();  
    public static volatile Queue<HashMap<String, String>> validinput = new LinkedList<HashMap<String, String>>();  
    public static volatile Queue<HashMap<String, String>> filteredinput = new LinkedList<HashMap<String, String>>();  
}
```

5.2.2 Threads

Our threads do not directly communicate with each other, they all communicate through the aforementioned queues. To always have the latest queue however, we have made all threads synchronized to keep up to date with all changes that happen within the queues.

```
/**  
 * Runs the thread  
 * Gives the data a timestamp and stores it in ./data  
 * Filename is "stationnumber - timestamp.json"  
 */  
@Override  
public synchronized void run() {  
    System.out.println("[DATA STORAGE] Running " + threadName);  
  
    // Init: "all data" folder  
    File directory = new File(defaultstoragelocation);  
    if (!directory.exists()) {  
        directory.mkdir();  
    }  
}
```

6 Server configuration

6.1 Installation, info, and elaboration on choices

- Management server

The management server provided by the Hanzehogeschool has been slightly modified to accommodate for internet through a reverse proxy. The program Privoxy has been used to provide the means of downloading and installing new packages for server B and server C.

- Server A

- OS: Default windows
- IP: 10.0.9.2
- Hostname: windows
- Use case: Collecting and redirecting data to the java application.
- Sidenotes: Houses the weather generator.

Server A was provided by the Hanzehogeschool and is used for housing the weather generator. Providing all data for the entire project. This is the server's only purpose.

- Server B

- OS: Ubuntu server 20.4
- IP: 10.0.9.3
- Hostname: java
- Use case: Collecting data with port 7789 from all IP addresses.
- Sidenotes: -

Server B has been configured to be as light-weight as possible. Therefore, we have installed the latest Ubuntu server (without GUI) version. Due to the limited drive space, this was just enough to install the OS and Java. The specific Java installation we have used is the jdk-11.0.10+9 because it contains the entire java-set. No further tweaks have been made to the OS or runtime environment. An Ubuntu Server is very lightweight. The application is compiled in a .jar file and run from **/mnt/Scraper.jar**.

- Server C

- OS: Ubuntu server 20.4
- IP: 10.0.9.4
- Hostname: data
- Use case: Stores data from java application and houses the web interface.
- Sidenotes: -

Server C has been configured to store data. This has been done through a Samba network share. Server B directly mounts this share and saves data on the share. This share houses all the json-data provided for the client. This server also houses a webserver: Apache. Apache has been configured to be as light-weight and featureless as possible to make sure all drive space is available for the actual data. To have user login functionality, the server also houses a small MariaDB database. This is only used for account data and location lookup.

6.2 Throughput of data

Due to the amount of data, if the server ever struggles with the amount of data sent, safety measures have been deployed. The queue sizes have a limit to how full they can be before they will halt the process until the queues have been emptied. Priority will be focused on client data (and not saving all data). And if an emergency ever occurs, new data will be ignored until all parts function again and have emptied their queues. This way we can guarantee that the throughput of information is always valid and no parts of the process get overwhelmed by the amount of data.

6.3 Transport of data

Transport of weather data is sent by a generator on server A to server B through an opened port. This client will open 800 connection which parse 10 files every second. Resulting in 8000 lines of new data each second.

Server B's Java application directly stores data on a Linux samba network share provided by Server C. Data is never stored on Server B as this server's only purpose is to run the java script.

6.4 Files / folders structure

The file structure has been arranged to contain all weather data in one place. This makes moving the files easier.

Naam	Gewijzigd op	Type	Grootte
UniversityTeheran	31-1-2021 12:21	Bestandsmap	
10030-1612052632122.json	31-1-2021 01:23	JSON-bestand	1 kB
10030-1612052632123.json	31-1-2021 01:23	JSON-bestand	1 kB
10030-1612052632124.json	31-1-2021 01:23	JSON-bestand	1 kB
10030-1612052634925.json	31-1-2021 01:23	JSON-bestand	1 kB

The file names are structured as **WEATHERSTATIONID-UNIXTIMESTAMP**, this has been done to easily query on specific weather stations and compare the timestamps to the current time. This way you can query easily through the files the application generates.

The json-file itself is structured as a single line consisting of all data:

```
{"Storm": "false", "Temperature": "-12.7", "StationLevelPressure": "1001.2", "Snow": "true", "Windspeed": "21.8", "CloudCoverage": "72.9", "Time": "00:23:52", "DewPoint": "-16.8", "Timestamp": "1612052632122", "Date": "2021-01-31", "Rain": "false", "Precipitation": "0.00", "SnowDrop": "0.3",
```

If you would structure this json-file to a more readable format:

```
1  {
2    "Storm": "false",
3    "Temperature": "-12.7",
4    "StationLevelPressure": "1001.2",
5    "Snow": "true",
6    "Windspeed": "21.8",
7    "CloudCoverage": "72.9",
8    "Time": "00:23:52",
9    "DewPoint": "-16.8",
10   "Timestamp": "1612052632122",
11   "Date": "2021-01-31",
12   "Rain": "false",
13   "Precipitation": "0.00",
14   "SnowDrop": "0.3",
15   "StationNumber": "10030",
16   "SeaLevelPressure": "1003.3",
17   "Visibility": "25.2",
18   "Tornado": "false",
19   "WindDirection": "32",
20   "Client": "[]",
21   "Freeze": "true",
22   "Hail": "false"
23 }
24
```

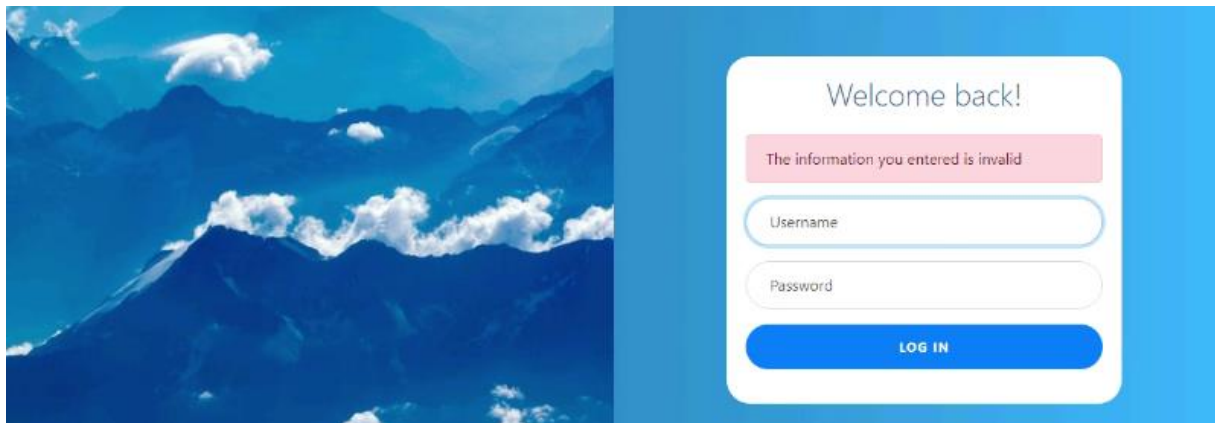
Specific info for a client would have their own client info and will be placed in their own folder:

```
"WindDirection": "314",
"Client": "[UniversityTeheran]",
"Freeze": "true",
"Hail": "false"
```

7 Data visualization (front-end)

7.1 Login

At the login webpage a user must fill in a username and a password to proceed on the website. If either of the two variables are not filled in or are invalid the user will be notified with an error message. Usernames and passwords are provided by RandomIT and stored in a database. Once the log in button is pressed, the username and password filled in by the user, are checked for validation. Once a login is successful the user is redirected to the homepage. No webpages, except for the login page, can be accessed by users who are not logged in.



7.2 Map

The homepage of the website is a map of the world with markers located at the relevant weather stations for the user. The user can click on the markers to see a small table with data collected by that station. At the marker a clickable link, called 'More data', will take the user to a webpage with all the data collected by that station.



7.3 Station information table

This webpage displays all the data collected by a specific weather station in a graph. The station collects the following datatypes:

- Date + Time.
- Temperature in °C.
- Dewpoint in °C.
- Precipitation in centimeters.
- Snowdrop in centimeters.
- Cloud coverage in percentage.
- Visibility in kilometers.
- Windspeed in kilometers per hour.
- Wind direction in degrees (0-360).
- Station level pressure in millibar.
- Sea level pressure in millibar.
- Freeze, rain, snow, hail, thunder, tornados as 'Yes' or 'No'.

On this webpage a button is visible which can be clicked on to download the data into an XML-file.

RandomIT development site Home Top 5 stations Welcome admin

914900 CHRISTMAS ISLAND, NEW ZEALAND

Download XML

Date	Storm	Temperature	Air Pressure	Snow	Windspeed	Cloud Coverage	Dew Point	Rain	Precipitation	Snow Drop	SeaLevelPres
2021-02-01 16:23:31	No	25.7°C	1006.6mbar	No	25.1km/h	76.6%	19.7°C	No	0.00cm	0.0cm	1006.6mbar
2021-02-01 16:23:31	No	25.7°C	1006.6mbar	No	25.1km/h	76.6%	19.7°C	No	0.00cm	0.0cm	1006.6mbar
2021-02-01 16:23:31	No	25.7°C	1006.6mbar	No	25.1km/h	76.6%	19.7°C	No	0.00cm	0.0cm	1006.6mbar
2021-02-01 16:23:30	No	25.7°C	1006.6mbar	No	25.0km/h	76.6%	20.2°C	No	0.00cm	0.0cm	1006.6mbar
2021-02-01 16:23:30	No	25.7°C	1006.6mbar	No	25.0km/h	76.6%	20.2°C	No	0.00cm	0.0cm	1006.6mbar

We also implemented a table with contain the top 5 stations based on the peak temperature of that day.

RandomIT development site Home Top 5 stations Welcome admin

Top 5 of weather station (Based on peak temperature)

Station	Date	Storm	Temperature	Air Pressure	Snow	Windspeed	Cloud Coverage	Dew Point	Rain	Precipitation	Snow Drop	SeaLevelPres
404770 JEDDAH INTL, SAUDI ARABIA	2021-02-01 16:14:53	No	22.2°C	1022.1mbar	No	12.1km/h	39.4%	13.3°C	No	0.00cm	0.0cm	1016.6mbar
404950 SULAYEL/ASSULAYYIL, SAUDI ARABIA	2021-02-01 15:57:27	No	19.0°C	1044.2mbar	No	18.5km/h	97.6%	-1.6°C	No	0.00cm	0.0cm	1016.6mbar
404450 AL-KHARJ, SAUDI ARABIA	2021-02-01 16:14:52	No	18.6°C	1019.2mbar	No	28.2km/h	10.5%	12.3°C	No	0.00cm	0.0cm	1016.6mbar
404480 DUBAI, UNITED ARAB EMIRATES	2021-02-01 16:14:52	No	16.6°C	1053.2mbar	No	23.1km/h	62.4%	9.3°C	No	0.00cm	0.0cm	1016.6mbar
404980 BISHA, SAUDI ARABIA	2021-02-01 16:14:55	No	15.9°C	993.8mbar	No	7.9km/h	93.5%	3.7°C	No	0.00cm	0.0cm	976.6mbar

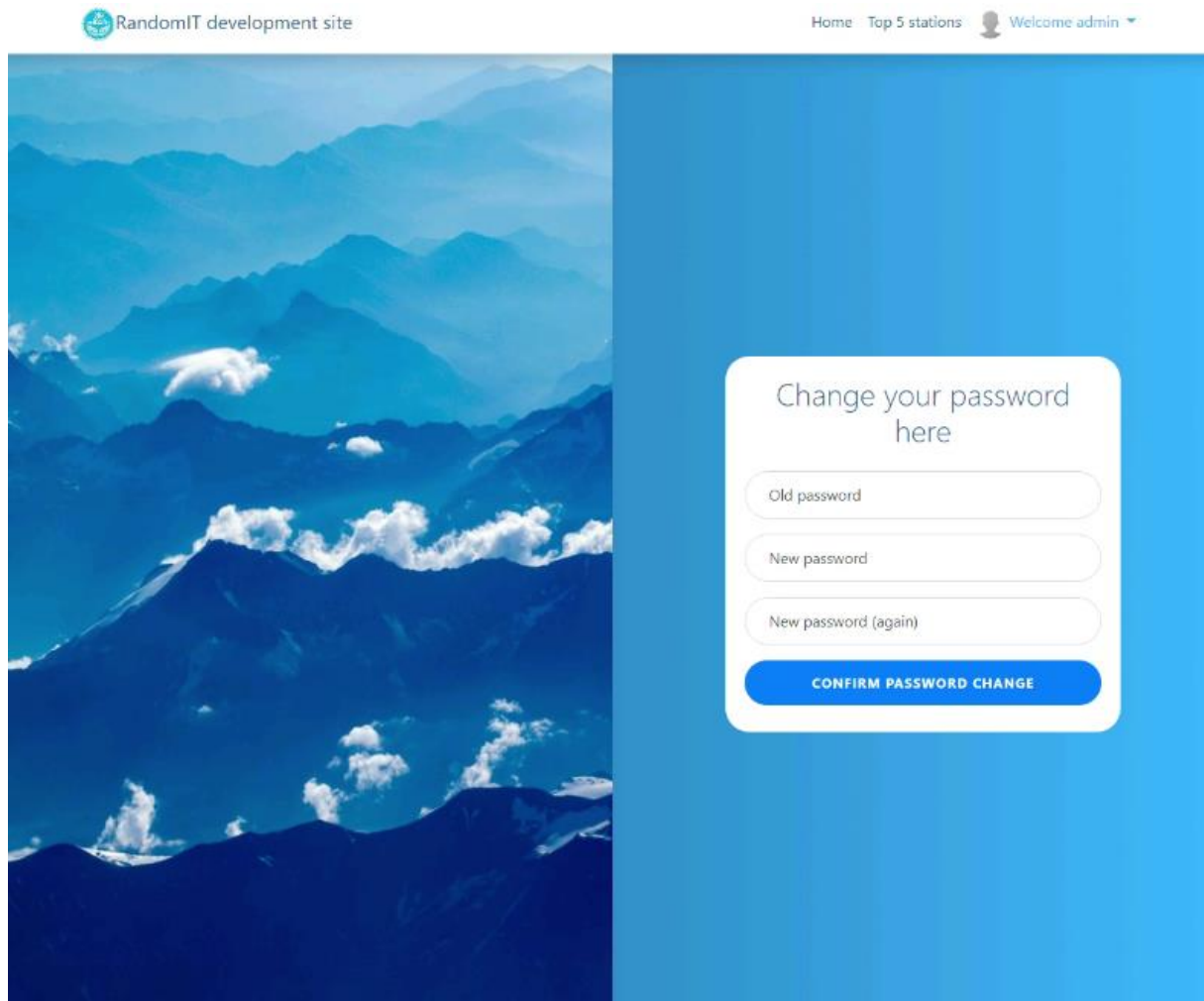
7.4 Downloading data

Once a user clicks the download button on the station information page an XML-file will be downloaded to the user's device. This XML-file contains all the data collected by the station which is available in the table. The XML-file will be named after the stations number.

```
<?xml version="1.0"?>
- <WEATHERDATA>
  - <MEASUREMENT>
    <DATE>2021-02-01</DATE>
    <TIME>12:22:14</TIME>
    <TEMP>17.4</TEMP>
    <DEWP>12.7</DEWP>
    <PRCP>0.00</PRCP>
    <SNDP>0.0</SNDP>
    <CLDC>71.2</CLDC>
    <VISIB>20.7</VISIB>
    <WDSP>22.8</WDSP>
    <WNDDIR>237</WNDDIR>
    <STP>1006.6</STP>
    <SLP>1006.6</SLP>
    <FREEZE>false</FREEZE>
    <RAIN>false</RAIN>
    <SNOW>false</SNOW>
    <HAIL>false</HAIL>
    <STORM>false</STORM>
    <TORNADO>false</TORNADO>
  </MEASUREMENT>
  - <MEASUREMENT>
    <DATE>2021-02-01</DATE>
    <TIME>12:22:14</TIME>
    <TEMP>17.4</TEMP>
    <DEWP>12.7</DEWP>
    <PRCP>0.00</PRCP>
    <SNDP>0.0</SNDP>
    <CLDC>71.2</CLDC>
    <VISIB>20.7</VISIB>
    <WDSP>22.8</WDSP>
    <WNDDIR>237</WNDDIR>
    <STP>1006.6</STP>
    <SLP>1006.6</SLP>
    <FREEZE>false</FREEZE>
    <RAIN>false</RAIN>
    <SNOW>false</SNOW>
    <HAIL>false</HAIL>
    <STORM>false</STORM>
    <TORNADO>false</TORNADO>
  </MEASUREMENT>
  - <MEASUREMENT>
```


7.5 Change password

Users can change their password when they go to the 'Change password' webpage. On this page the user needs to fill in their current password and fill in a new one. They also need to fill in the new password a second time to confirm it is typed correctly. Once the user confirms the change and click on the 'Confirm password change' a database query will change their password to their newly filled in password.



RandomIT development site

Home Top 5 stations Welcome admin

Change your password here

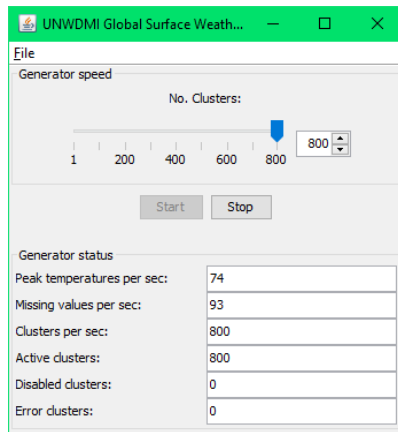
Old password

New password

New password (again)

CONFIRM PASSWORD CHANGE

8 Stress testing



As shown in the screenshot, the application can run 800 simultaneous connections that are being connected to the data receiver thread through the socket.

```
Rawinput size: 2132
Validinput size: 24966
Filteredinput size: 16000
All succeeded 597| All discarded = 0| Data for client: 10| clientdata succeeded: 10| Abandoned: 0| Abandoned IO: 0
```

The Rawinput is the incoming data from the generator.

The Validinput is the data that has been parsed.

The Filteredinput is the data that has been filtered according to the clients necessities.

The screenshot shows a file explorer window with the following table of files:

Name	Date modified	Type	Size
UniversityTehran	2/1/2021 11:15 PM	File folder	
20130-1612217701117	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701124	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701131	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701138	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701151	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701157	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701164	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701171	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701178	2/1/2021 11:15 PM	JSON Source File	1 KB
20130-1612217701190	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701522	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701529	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701536	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701543	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701557	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701564	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701571	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701578	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701585	2/1/2021 11:15 PM	JSON Source File	1 KB
20860-1612217701599	2/1/2021 11:15 PM	JSON Source File	1 KB
22690-1612217691370	2/1/2021 11:14 PM	JSON Source File	1 KB
22690-1612217691375	2/1/2021 11:14 PM	JSON Source File	1 KB
22690-1612217691380	2/1/2021 11:14 PM	JSON Source File	1 KB
22690-1612217691385	2/1/2021 11:14 PM	JSON Source File	1 KB

Eventually the data will be stored in JSON format and the requested data by the client will be stored in a separate folder.