

UNWDMI  
Team Schnitzel

Project report

Theme 2.2  
February 2017  
332975 - Bart Bakker  
346877 - Danny D. Jager  
347419 - Eran B. Machiels  
274633 - Joost P. de Vreede  
343567 - Jouke Y. Rienstra  
351509 - Matthijs S. Bonnema



# Title page

**UNWDMI – Team Schnitzel***Project report*

This document encompasses the information used for the production of, the design choices about and the instructions for the use of the *Team Schnitzel Weather Service.*

This document is part of project for Theme 2.2 of the HBO-ICT education of the Hanzehogeschool Groningen.

Team members:

332975 Bart Bakker

346877 Danny D. Jager

347419 Eran B. Machiels

274633 Joost P. de Vreede

343567 Jouke Y. Rienstra

351509 Matthijs S. Bonnema

Tutors:

General tutor A.E.A.G. Thüss

Technical tutor J.H.F. Bredek

*February 2017*

# Abstract

This document gives a guide to all the code produced for the project of theme 2.2 of Infrastructures. The project will be discussed from the points of the client at first. What are the requirements? Then the solutions of how it was accomplished will be procedurally discussed.

# Table of contents

[Title page I](#_Toc474411381)

[Abstract III](#_Toc474411382)

[Table of contents V](#_Toc474411383)

[Introduction 1](#_Toc474411384)

[I. About the United Nations Weather Data Management Institute 3](#_Toc474411385)

[II. Functional design 5](#_Toc474411386)

[Customer requirements 5](#_Toc474411387)

[Definition of the Balkan 5](#_Toc474411388)

[Analysis of data 7](#_Toc474411389)

[Design Mockups 8](#_Toc474411390)

[III. Technical design 9](#_Toc474411391)

[Project management 9](#_Toc474411392)

[Version control 10](#_Toc474411393)

[Communication tools 10](#_Toc474411394)

[Programming languages 10](#_Toc474411395)

[Dataprocessing 11](#_Toc474411396)

[OS choices 11](#_Toc474411397)

[Raspberry Pi 11](#_Toc474411398)

[Virtual server 11](#_Toc474411399)

[IV. Execution of the project 13](#_Toc474411400)

[HTTPS 13](#_Toc474411401)

[File Sharing 13](#_Toc474411402)

[Firewall 13](#_Toc474411403)

[Fail2ban 13](#_Toc474411404)

[Data parsing 13](#_Toc474411405)

[Server.py (port number) 13](#_Toc474411406)

[Extrapolate.py 14](#_Toc474411407)

[Init.py True/False True/False 14](#_Toc474411408)

[Balkan\_filter.py 14](#_Toc474411409)

[Country\_filter.py 14](#_Toc474411410)

[Creating a database seeder 14](#_Toc474411411)

[Using data offline (CSV download) 15](#_Toc474411412)

[Top 5 visibility of the Balkan area 16](#_Toc474411413)

[Windows, Linux difference 18](#_Toc474411414)

[Authentication Laravel 19](#_Toc474411415)

[CSV to Array 19](#_Toc474411416)

[Receiving the humidity data 20](#_Toc474411417)

[The logic behind the (updating) humidity graphs 21](#_Toc474411418)

[When the window is loaded 21](#_Toc474411419)

[When the selectbox changes 21](#_Toc474411420)

[Drawing the graph 22](#_Toc474411421)

[Updating the graph 23](#_Toc474411422)

[V. Conclusion 25](#_Toc474411423)

[VI. References 27](#_Toc474411424)

# Introduction

Team Schnitzel (TS) is a part of the United Nations Weather Data Management Institute (UNWDMI). The team produces applications for clients of the UNWDMI. In the beginning of January 2017, the team was contacted by a client from Serbia, the University of Belgrade. The University was interested in the opportunities the UNWDMI could offer them. At a meeting in mid-January, it was decided that TS would produce an application to help the University with their research.

During the following two months, TS worked out their application. It is called the ‘*Team Schnitzel Weather Service*’ (TSWS). The TSWS receives localized weather data from weather stations from all over the world. The data is received via a collection server. Currently, this server is set up to receive up to a maximum of 800 weather stations at the same time. For the further use of this data, a distribution server is used. This server provides the clients with the requested data for their research.

Both servers are lacking resource overhead with the use of these applications. The resource allocation provided quite a few difficult design choices. But the provided TSWS still provided the data as requested by the client.

This report provides the background information of the whole production process of the TSWS. In the first chapter, the general information about the UNWDMI will be discussed. What is the UNWDMI, what is the data that is available to clients? The second chapter will provide information about the case of the University of Belgrade. What are the wishes of the clients and how do they want to access this data? In the third chapter, the technical design will follow. The fourth chapter will discuss all the technical implementations about the solution. In the fifth chapter, a short conclusion will follow.

# About the United Nations Weather Data Management Institute

The United Nations Weather Data Management Institute (UNWDMI) is an organization that provides global weather information for the use of local governments, regional weather forecasting bureaus and other organizations that have an interest in the provided data. It has its headquarters in Groningen, the Netherlands. It is a small organization of 90 people that receives weather data from all local weather institutes from countries connected to the United Nations.

|  |
| --- |
| C:\Users\Joost\AppData\Local\Microsoft\Windows\INetCacheContent.Word\UNWDMI_Global.png  Figure 1 – A rendering of the connected UNWDMI weather stations, across the globe. |

These local institutes provide weather data from their own countries to the UNWDMI in a standardized manner. In all, there are 8000 stations, all over the world. Figure 1 shows a representation of the locations over the globe.

The bulk of these stations lays within Europe, Northern America and Asia. All data stations provide an update every second. The UNWDMI collects this data in a centralized system. After the collection, the data will be used for applications built to the specific needs of its customers.

The UNWDMI currently has two systems to its disposal for these two tasks. Both systems are chosen to make optimal use of the available hardware, regarding the amount of data that needs to be processed. This means that every bit of processing power available in the system will be used to the fullest. The system will always produce the most accurate result possible.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Team member*** | ***Department*** | | *Matthijs S. Bonnema* | Management | | *Joost P. de Vreede* | Service Management | | *Bart Bakker* | ICT Services | | *Jouke Y. Rienstra* | Application Development | | *Eran B. Machiels* | General Management | | *Danny D. Jager* | Data Acquisition |   Table 1 – The members of Team Schnitzel and their departments within the UNWDMI. |

Team Schnitzel (TS) is one of several production teams within the UNWDMI. The production team is a team of people from several people from departments within the UNDWMI. The current consistency of TS is displayed in table 1. Since all teams within the UNWDMI consist of several departments, expertise from all their working fields will contribute to a more streamlined product.

# Functional design

## Customer requirements

Team Schnitzel (TS) was contacted by a delegation from the University of Belgrade. The University expressed the need for weather data for the from the regions in which they perform their primary research. In the start of January, the delegation visited the headquarters of the UNWDMI for a meeting with TS, to work out the fine details of their expected uses.

The main points of their research encompass climate changes in short to medium timeframes. It is the goal to be able to monitor trends within all the different datasets the UNWDMI has to offer. The University also asked for the calculation of two specific datasets:

* A calculation of the humidity values for all the weather stations within Serbia. The data should be available at intervals of 10 seconds per station. The data should be represented in a graph for the past 60 minutes. It would be valuable if it is possible to zoom in within the graph, so that even subtle changes are easy to recognize. The primary goal for these values is to help academics in the research for fungi and other trails involving humidity dependent variables.
* A representation of the 5 weather stations, within the Balkan area, that have the highest visibility distance. The visibility ranking should be calculated per day, from midnight onwards. A permanent history of these values should be available at all times. If the data for the current day is requested, the values should be calculated at once for all the measurements until that point of the day.

An interface needs to be created for all of the users within the University to interact with the data. A website would suffice, but there are certain precautions to take into account for the design. Most of the researchers in the field will use their mobile devices to access the data. This could be tablets, as well as ‘smartphones’. The website should look the same on all devices, regardless of the physical size or screen resolution. Also, in some areas where the academics operate, mobile coverage is poor or non-existent. It should be possible to access the most recent data beforehand. Most of the devices that will be used, are able to work with spreadsheets.

The whole system should be protected against unauthorized access. The University expressed concern that their data should not be available to users that have not been granted permission from them. It was decided to protect the interface with a login system. TS will manage the user accounts and passwords. All updates within the users will be submitted to TS by the university on a regular basis.

All returned results in the website, together with the saved data in the underlying systems should be stored in the local time formatting of Europe/Belgrade.

## Definition of the Balkan

During the development of the application, it became apparent that there are several possibilities to define the area that is colloquially called ‘the Balkan’. The three most logical uses were conveyed to the University. These versions were as follows:

1. The Balkan peninsula, containing parts of (but not limited to) Croatia, Greece, Serbia, Slovenia and the whole of Albania, Bulgaria, Bosnia and Herzegovina, Kosovo, Macedonia and Montenegro. This set contains 138 weather stations.
2. The states which have their geographical borders beyond the Balkan peninsula. These states are: Albania, Bosnia and Herzegovina, Bulgaria, Croatia, Greece, Kosovo, Macedonia, Montenegro, Romania, Serbia, Slovenia and Turkey. This set contains 317 weather stations.
3. The same countries as in set 2, except Greece and Turkey. This set contains 196 weather stations.

The reply from the University was to use the first set of weather stations for the visibility calculations. Figure 2, 3 and 4 represent the coverage of these possibilities.

|  |
| --- |
| Figure 2 – A rendering of the all weather stations within the Balkan peninsula. |

|  |
| --- |
| Figure 4 – A rendering of all weather stations within the Balkan area, according to state borders, excluding Turkey and Greece. |

|  |
| --- |
| Figure 3 – A rendering of all weather stations within the Balkan area, according to state borders. |

## Analysis of data

|  |
| --- |
| Figure 5 – A selection of an incoming data stream. |

The data of all weather stations is provided to the UNWDMI in a standardized manner. It is transmitted in a byte stream, containing an XML structure. A single set of data contains information about 10 stations at the same time. This data needs to be translated to a format that can be used by all the other systems of Team Schnitzel Weather Service.

The original data string is always compiled in the same manner. Figure 5 shows an example of the received data. It is set up in an XML format. This is a way of dividing large amounts of data into logical segments. It is a structure that can use multiple *layers* of data. All layers and segments start and end with a corresponding *tag*. A tag is a string that is encompassed by the < and > signs. Furthermore, the last of the two tags always has a forward slash ( / ) in front of it. This is called the *closing tag*.

All data streams start and end with the *WEATHERDATA* tag, as shown on line 1 and 19 in figure 5. In between these tags are the data sets from the 10 stations. Figure 5 only shows the data from one station, between the *MEASUREMENT* tags on lines 2 and 17. These lines are repeated another 9 times, each time for a different station. The data provided by the stations are as follows (the number represents the line in figure 5):

1. The station number. All weather stations have a corresponding number that identifies the location and other information that is specific to the location.
2. The date on which the data was sent.
3. The time on which the data was sent.
4. The temperature in °C.
5. The dew point in °C.
6. The air pressure at the height of the weather station, in millibar.
7. The air pressure at sea level, in millibar.
8. The visibility in kilometers.
9. The wind speed in kilometers per hour.
10. The precipitation in centimeters.
11. The amount of fallen snow, in centimeters.
12. A string that represents events that happened since the last data stream. It represents the following six possible events: freezing, downfall as rain, snow and hail, lightning, tornado’s and whirlwinds.
13. The cloud coverage as a percentage.
14. The wind direction in degrees.

## Design Mockups

These illustrations were the primary inspiration for the final product. These mockups provided a basis to transpose into a functioning website design.

|  |
| --- |
| Figure 6 – Mockup: Login screen. |

|  |
| --- |
| Figure 7 – Mockup: Home screen. |

|  |
| --- |
| Figure 8 – Mockup: Humidity station selection. |

|  |
| --- |
| Figure 9 – Mockup: Humidity graph. |

|  |
| --- |
| Figure 10 – Mockup: Top 5 graph. |

# Technical design

This chapter describes the how the project was planned to be executed. The technical specifications and ideas to get all functions working are discussed The first part details the collaborative platforms of Team Schnitzel, and in the latter half all technical details for the actual application are detailed.

## Project management

This project was realized with the utilization of a Scrum method. This was done online via the collaboration platform *Trello.com*. It provides a digital representation of a real scrum board. All members can receive updates automatically and update everything dynamically.

Like with all Scrum methods, the work was divided in *sprints*. Each sprint lasted one week.

The following Trello states were used for this project:

* Backlog: Idea’s for the project
* Todo: Parts of the project that have to be executed by a team member.
* Doing: When someone is working on a certain part of the project.
* To be tested / merged: When code has to be merged between different team members or when code has to be tested.
* Done / Tested: Parts that are finished.

Figure 11 shows the Scrum board.

|  |
| --- |
| Figure 11 – A screen capture of the TS Scrumboard, via Trello.com. |

## Version control

TS has a good experience with Git as version control for software development. Git shows historical changes by different team members in the code and makes it possible to merge files more easily together.

GitHub was used, which is a visual presentation of Git. A private repository was used for this project so that the code is not shared with outsiders.

|  |
| --- |
| Figure 12 – A selection of updates within GitHub. |

## Communication tools

For communications, WhatsApp was used for quick interactions. Meetings and e-mail were used for formal discussions. Trello was used to communicate and recap the information for specific points in the production of the application.

## Programming languages

For the realization of this project several programming languages were used.

For the user interface, a website was used which was programmed in PHP. It is conceived in the *MVC* framework *Laravel*. Laravel is object oriented and has several useful functions. MVC stands for Model View Controller. This means that programming logic is divided.

The model contains interactions with the database. The view is a visual representation of the data and the controller contains all the logic.

Larvel uses a file for routes (url). A route could be /humidty/stations. Which could provide all the stations from Serbia from example.

For the parsing of the data to .csv files, Java and Python were used. After it’s programmed, it will be decided which one is running faster and can processing the data the best.

## Dataprocessing

Since the data processing will mainly take place on the Raspberry Pi and TS will have limited resources. TS will use a folder structure to save the data. A database will probably not be able to process 8000 updates per second. All the data will be saved in the folder with the current day and a file name that uses the station number. This way TS will be able to filter and delete the data easily when needed.

Since TS is limited in resources, the data can only be stored for a couple days. After 3 days, the data should be deleted, to save up space for the next day.

TS will use a database to save the visibility averages, since this has to be a rolling calculation for the current day, and easily accessible for the website. These calculations only will occur when someone requests the, this way there won’t be unnecessary load on the server.

## OS choices

Since some members of TS have a lot of experience with certain operating systems, we made the following OS choices;

### Raspberry Pi

For the Raspberry Pi we will use openSUSE 42.2 Leap. The provided operating system, Raspbian Jessie, is a 32-bit operating system. This OS does not make use of all the capabilities of the 64-bit CPU on the Raspbery Pi. openSuse is 64-bit and therefore a better OS to pick. Unfortunately, openSUSE is compared to similar linux distributions, harder to use. Therefore, TS has to put more effort in the Raspberry Pi then expected in the first place. But the performance TS will gain, is worth it.

### Virtual server

The server will be used to save the data and to serve a website. Since there are limited resources available, TS will use a lightweight OS. In this case CentOS will be used. TS is very familiar with this OS and therefore setting up the server doesn’t take a lot of time. CentOS is an open enterprise server operating system. Therefore, it is optimized for running Apache and other solutions.

The server will be running version 7 of CentOS, since version 6 does not support PHP 7 natively. And the website will make use of Laravel, which requires PHP 7. Also, MariaDB is better supported on CentOS 7, which we will use to store the login info for the users and to store the average visibility.

# Execution of the project

In this chapter, all the implemented technologies for the actual application will be discussed.

## HTTPS

|  |
| --- |
| Figure 13 – The representation of the working SSL certificate, in the web browser Chrome. |

To secure the website a SSL certificate is used. This certificate is made using the Let’s Encrypt certbot. This “bot” is written in Python and makes installing a SSL certificate really easy. All the web traffic to the server gets directed to https (the SSL connection)

## File Sharing

Since an SD Card would, besides not being reliable, be too slow to write all the station files too. Therefore, TS will mount a folder of the virtual server on the Raspberry Pi. This is done using NFS4, during testing we concluded NFS is the best protocol, since it is good at transferring lots of smaller files at the same time. Version 4 is used since it encrypts the connection.

Whenever the Raspberry Pi boots, it automatically mounts the folder, using the following cronjob:

*@reboot mount 145.33.225.145:/home/share /root/WeatherStation/data*

On the server, the firewall only allows the IP-address of the Raspberry Pi to connect, therefore no one else would be able to retrieve the data.

The website can access the data of the stations using a symbolic link to the folder in the /home/ directory.

## Firewall

To make sure the server and the Raspberry Pi are not vulnerable for the outside, the firewall only lets the 2 hosts connect to each other on the desired ports. Also, a couple ports are open for the outside, for example the default port of the generator.

## Fail2ban

Since SSH is open for the outside during development, TS installed fail2ban. It is configured so every IP-address can enter the login details 3 times. After these 3 times the IP gets blacklisted and won’t be able to connect for a certain time.

## Data parsing

The data processing takes place mainly on the Raspberry Pi. TS wrote a XML parser in Python and in Java. TS started writing the parser in Python, because one of the team members of TS was able to write the parser in Python. Python is good at calculating missing values, which sometimes occur during generation or transmission.

### Server.py (port number)

Since a database would not be able to process all the 8000 entries per second, TS made the decision to use a flat-file system. The parser filters all the unnecessary incoming data and saves it to a csv file. Csv files are easy to use and only contain the necessary data. This way TS is able to save the data on just 10 GB of hard drive.

TS wrote a manual parser. The existing parser libraries would be too resource intensive for the Raspberry Pi.

|  |
| --- |
| Figure 14 – The folder structure with the filters. |

The parser saves the data per station in a folder with the current date. This way the data is neatly organized and easy to filter. The client requested to store the data for 2 days. We save the data for one extra day, to make sure there is always enough data to calculate the averages.

### Extrapolate.py

|  |
| --- |
| Figure 15 – The .csv files per station. |

The generator also generates on purpose missing values. These values are noticed during the calculation of the humidity. When there are missing values, the whole dataset gets given to function that extrapolates the data. It uses Numpy (a Python library) to polyfit a curve on it. This curve gives the expected next value. When there is not enough data to calculate a proper curve, the parser uses the last data point.

### Init.py True/False True/False

Every day, at 23:55 a script runs that creates the folder for the coming day, it also creates the csv files with a header for all the stations. After this script, it deletes the folder and it contents of 3 days in the past. Both parameters are optional, the parameters are for use without the interactive command line. Inserting the first True, automatically creates the folders and stations files for the next day. The second True deletes folder with the date of 3 days in the past.

### Balkan\_filter.py

Since the Balkan is a non-definable area, TS made a script that fills the filter with a list of stations located in the desired Balkan area. This is an interactive command line script.

### Country\_filter.py

This script creates a filter with the countries you desire. It can append to the existing filter list.

|  |
| --- |
| Figure 16 –The run() method of the UsersTableSeeder class. |

## Creating a database seeder

Because an authentication process was a requirement for the application, TS needed to have some login information for their local work environment. The easiest way is to let everyone in the project group have an easy way to work with test accounts without the having to create one manually. Laravel, the implemented MVC framework, already has a login system built-in. The proces of creating test accounts is simplified with the concept known as database seeders. With the PHP Artisan CLI a seeder class can easily be created. The Seeder class must have a *run()* method in order to work correctly.

Figure 16 shows the implementation of this code. As shown, the above 5 rows get inserted in the users table. The details of the account are always called test@domain.com and secret with the current loop counts as a suffix. The passwords are hashed with the *bcrypt* (blowfish) algorithm.

*Example:*

*Username: test2@domain.com*

*Password: secret2*

|  |
| --- |
| Figure 17 –The run() method of the default DatabaseSeeder class. |

However, before the “*php artisan db:seed*” command can be run, the recently made class (*UsersTableSeeder*) must be specified in the DatabaseSeeder class by using the *call()* method.

The project members can now run a simple “*php artisan db:seed*” command in the command line and the test accounts are generated for their local work environment.

Sources:

*/database/seeds/DatabaseSeeder.php  
/database/seeds/UsersTableSeeder.php*

## Using data offline (CSV download)

|  |
| --- |
| Figure 18 – Humidity view with the ‘Download current data to csv’ button. |

One of the requirements for the project was the ability to use data offline so it can be used in the field. This paragraph contains the explanation of the implementation.

For example, we take the humidity view. This view contains the humidity data of the last 60 minutes. The view contains a download to csv button. This button uses the following URL route /humidity/{id}/download.

The ID of the station is known within the page. When the Serbian station Palic is selected, it uses the corresponding ID in this case 130670 so the URL becomes /humidity/130670/download.

As explained earlier in the paragraph ‘receiving the humidity data’. In the controller, which contains all the logic, we read a file and get all the data in an array. For the presentation the array is converted to JSON and loaded with JavaScript. For the download to csv we use the array and wrote a function which parses this array to a comma separated csv file. The route /humidity/{id}/download calls the downloadData() function of the Humidity controller.

|  |
| --- |
| Figure 19 – The code needed for the csv download |

|  |
| --- |
| Figure 20 – The contents of a downloaded CSV file |

In this function. The right headers are set so it doesn’t output on the page. An array is received with the calculateData($id) function. The fields that we want in the headers are set for example: stn, time, humidity. Every value in the array is then converted to a row with a comma separation.

When the conversion is done, we close the output fclose($output); The CSV is downloaded.

## Top 5 visibility of the Balkan area

In this paragraph, we explain how the top 5 visibility of the Balkan area is implemented.

All the logic of the view is implemented in the Top5Visibility controller. Because there is so much data and we want to view it as fast as possible we make use of excel files. In this certain view we decided to calculate an average for the previous days and add this in a database because there wouldn’t be any changes anymore. The current day is calculated again because there can be new added data. Because of this feature the previous days load a load quicker which improves the user experience.

A couple of routes are registered;

* /top5visibility/{requestDate?} calls the home() function
* /top5visibility/{date}/live/data calls the getData() function
* /top5visibiliy/{date}download calls the downloadData() function.

**Home() function**

The home function in the controller requires a requestDate variable. This date is checked if it contains data and is converted and checked for the right time format. The time format we decided earlier is yyyy-mm-dd. We call the calculateData() function and calculate the data for a certain date. The result is parsed to the view. Afterwards the view decides what to do with the information.

**getData() function**

This function calls the calculateData() function and convert the resulted array to a JSON format. The JSON data is outputted at top5visibility/live/data. Because of this format we can easily read the data in the view with javascript.

|  |
| --- |
| Figure 21 – The splitting of the data. |

**calculateData() function**

This function receives the data from a certain day by reading a csv file from the disk. It reads the filenames based on the time and date structure. Because the directories are structured in a yyyy-mm-dd structure we can easily open the right files which saves a load of loading time. All the files that correspond to the selected data are opened and parsed.

The parsing reads every new line and adds the visibility of this line to the visibility variable. A counter is also updated.

|  |
| --- |
| Figure 22 – The insertion of the data into the database |

When the parsing is done the data is implemented in a database table called ‘Avarage visibility’.

|  |
| --- |
| Figure 23 – The final top 5 chart. |

**downloadData() function**

Parses data from a certain date to a csv file. The logic is explained in another paragraph.

**The view**

The view contains a visualization(bar chart) that shows the data that is calculated.

For the more technical readers out there u can view the source code. The code is also well documented.

The logic is available at:

*/website/app/Http/Controllers/Top5visibilityController.php*

The view is available at:

*/website/resources/views/top5visibility.blade.php*

## Windows, Linux difference

Because Linux and Windows use different code for line breaking we implemented a function that checks which OS is running. Based on the OS running it changes some code parts so it works om multiple systems.

## Authentication Laravel

To make the realization of the authentication process in Laravel easy, Team Schnitzel used the ‘built-in authentication’ functionality of Laravel. The reason Team Schnitzel chose to use the pre-built function of Laravel is because user authentication is already a solved problem. The built-in function has been tested and used by many developers and it saves a lot of time which can be used to realize the other requirements of the customer.

|  |
| --- |
| Figure 24 – The login screen for the application. |

Laravel has an easy scaffolding technique which generates a pre-built controller, middleware, views and even a basic layout. This is done by executing *‘php artisan make:auth’* in the command line.

|  |
| --- |
| Figure 25 –This constructor provides the start of the authentication method. |

One of the requirements was that users should not be able to register an account by themselves, so the registration form is hidden and only accessible by administrators.

There are multiple ways to prevent other pages from being accessed by guest visitors. One would be to group the routing in an ‘*auth middleware*’ (created by *php artisan make:auth*) and the other way would be to put the ‘*auth middleware*’ in the constructor of a controller where the current routing refers to. Team Schnitzel went with the latter option.

## CSV to Array

Because the CSV to array conversion plays a big part in this project, it’s important to explain this process very globally with a little bit of detail. The CSV to array conversion can be found in *HumidityController.php* and *Top5visibilityController.php*. The code is documented very well, so feel free to look in the source code when searching for more details.

The code starts with an if statement, where the code checks if a certain folder and file exists. This folder has an Y-m-d format. If the .csv file with the given id exists, it will execute the code within. The id of the file is the id of the weather station. For the humidity data, this gets handled by the HTML select box at /humidity where the name of a weather station is linked to their weather station id. The id then gets passed to JavaScript and an AJAX script calls the humidity/live/data/{id} page and expects a JSON response.

When the if statement passes, the code will then get the .csv file. Depending on the operating system, the data of the csv gets split up into an array. This is done by the php *explode()* function.

The measurement types (date, time, humidity, temperature, etc) then gets split up as well. The measurement types then become keys with an array as a value.

|  |
| --- |
| Figure 26 – Code that splits the csv format to a php array. |

The array is now a multidimensional array. The code now loops over the entire dataset, skips the first line, skips empty rows and when the data is valid, it gets split by a comma again. The data then get their specific measurement type assigned.

|  |
| --- |
| Figure 27 – The final array gets its data filled. |

An example of the final array data would be:

*$data = [  
 ‘temperature’ => [15.2, 16.3],  
 ‘humidity’ => [4.8, 7.2]  
];*

This is an example of the data, where $data[‘temperature’][0] is linked to $data[‘humidity’][0] (15.2 and 4.8) and $data[‘temperature’][1] links to $data[‘humidity’][1] (16.3 and 7.2)

Sources:

*/website/app/Http/Controllers/HumidityController.php  
/website/app/Http/Controllers/Top5visibilityController.php*

## Receiving the humidity data

The humidity data is received in .csv files from the Raspberry Pi. Because the humidity data, which is calculated on the Raspberry Pi must be received every 10 seconds, the first possible to solution that came to mind was to run a cron job on the server. This solution was implemented by developing a custom php artisan command. This command was supposed to be called every 10 seconds on the server using a Laravel scheduler (Laravel’s API for cron jobs).

Team Schnitzel later noticed that this was not a very smart decision because real-time data is always client-sided, so the next solution was using Javascript to call HTTP GET requests. Fortunately most of the logic could be reused, so not much time was wasted by changing the infrastructure.

The calculateData() function is where the data is received and filtered. For the more technical readers of this document: to view the calculateData() function in more detail, please refer to the file itself (can be found below). The code of the project is documented very well.

By accessing the url: /humidity/live/data/{id} (where id is the id of the datastation), the calculateData() function gets called and the data decodes to JSON. The next step is to use AJAX to render de data every 10 seconds and display it in the graphs.

|  |
| --- |
| Figure 28 – This code converts data to a JSON string. |

To explain the code above in a little more detail. When a user browses to a /humidity and clicks on a weather selection by making use of the HTML selectbox. The id of that weather station gets passed to AJAX. AJAX makes the call to /humidity/live/data/{id} where the function in the HumidityController.php gets called. $this->calculateData($id) is a private function within the same controller where all the logic gets handled. The array of data gets returned and is then decoded to JSON.

Sources:

*/website/app/Http/Controllers/HumidityController.php  
/website/app/Console/Commands/calculateHumidity.php (old: custom command)*

## The logic behind the (updating) humidity graphs

### When the window is loaded

When the window object is loaded, the codes within this function is being executed. This is done so the JavaScript won’t execute until the full page with all the elements is loaded. This saves loading time, so provides a faster website.

When the page is loaded, an Ajax script is being executed. Ajax stands for Asynchronous Javascript & XML. It dynamically shows the weather stations within Serbia, which reduces loading time. The JavaScript provides a good way to load them while you can’t see it loading.

The Ajax does a request to an url, which returns a JSON-object with all the stations. This is dynamically loaded in a dropdown menu. When you click on an item in the dropdown, the onChange() function is called.

### When the selectbox changes

When the selectbox changes, thus another station is requested to show his data, the onChange() function is called.   
When viewing a graph of a weather station, the graph dynamically updates the station every ten seconds. This is set by an interval.  
An interval is a sort of object, created every time a new interval is set. When you update the graph while there was another graph shown, you have to delete the existing interval. Otherwise you will have two intervals (one for

|  |
| --- |
| Figure 29 – The codes executed when the station is changed. |

the former shown graph and one when the new station is showing his graph) and a new interval for every change. See figure 29 for the actual code.

“Unset” means that all existing intervals are cleared. With other words, all the existing interval objects are removed. The new graph is shown, where argument “id” is the id of the requested station. When the graph is shown, a new interval is “set” with the id of the station.

Besides the updating of the graph, the download button is updated as well. The url of the download button is changed so it correspondents with the ID of the station. This is made so the current graph shown can be downloaded, even when you change the weather station.   
When downloading the data, the raw data that is shown in the graph is presented in a csv document.

### Drawing the graph

|  |
| --- |
| Figure 30 – The primary humidity graph. |

When the drawGraph() function is called, the graph is drawn. The application uses the functionalities of the graph library Highstock.   
This function does nothing more than making a new chart object on the page. This is also done in an Ajax request. A JSON object is returned with two keys, time and humidity, and values representing these keys. The data of time is loaded in the x-axis of the graph and the data of time is loaded in the y-axis. The data of the x-axis is converted to a number, so there won’t be any errors that some data isn’t a correct number.

Because there is a lot of data which is very hard to read, there is a zoom functionality at the bottom of the graph. With this functionality, you can zoom in to a specific area of the chart to view it in detail (see figure 31 for an example).

|  |
| --- |
| Figure 31 – The zoomed in humidity graph. |

When there is no data available for the graph, the script destroys the graph and shows an error message that there was no data available. This is removed again when there is a weather station chosen which actually has data.

## Updating the graph

Every ten seconds an interval executes a line of code. This code updates the graph, again via Ajax. The full dataset is fetched and the last point of that data set is accuired: both the last time value (x-axis) and the last humidity (y-axis) value.   
The time value is pushed to the categories, the values on the x-axis. The humidity value is added as a point on the graph, and “connected” to the newly added time point on the x-axis, because the graph checks the length of the data points and the length of the values on the x-axis. So when to both axes one value is added, the graph knows that they belong together.  
The function which adds the new point has some attributes that can be switched on or off. All attributes are switched on.   
When updating the graph, there is a new data point added to the right. Because the data is representing for the last 60 minutes, at the left a point is removed. So every time you look at the graph, the same amount of points is displayed. The updating happens with an animation: a new point is added and the line of the graph to that new point is drawn, it doesn’t just pop up. After that, the graph is redrawn, so you won’t notice anything when you are zoomed in. This is necessary because when the graph isn’t redrawn and just refreshes, the zoom is gone and the user has to zoom in again.

# Conclusion

The main goal of the project was to work out and visualize the requirements discusses in a meeting with our clients. Team Schnitzel had to discover what the client wanted and how they wanted that. These requirements had to worked out as good as possible, with our knowledge about software and networks. This visualization should be presented in an application.

The application provides a functionality to covert weather data generated by weather stations to visual data, readable by a user. At first, the data is generated as XML. On a Raspberry pi, the data is converted to a csv file, thereafter it is sent to the virtual machine.

This virtual machine is primarily used as webserver, so the data can be shown on a fully working website where the user can log in and logout. Connections between the server and the Raspberry Pi are secured with an SSL-certificate so nobody can intercept the data and use it for no good. Also the connection between the user and the server is secured with the same certificate, so user date is safe.

The client wanted two datasets. One of them was the humidity of all weather stations in Serbia. The graph should have shown all humidity points of the past hour, with an interval of ten seconds. After ten seconds, the graph should update (live) with the new data point.

The second dataset that had to be presented was a graph of the top five of the visibility of all the stations in the Balkan, where the stations with the highest visibility was number one. The timespan of this graph had to be from the time of watching the graph until 12:00 PM that night. No live updating was required there, only when refreshing the page.

Team Schnitzel completed almost everything the client required. The data is parsed to usable data on the server, the data is shown in a graph and the graphs are updated every ten seconds. “Almost completed” because the updating of the graph doesn’t work as our likings. It is not stable enough to say that is works like a charm, but the functionality is there. Furthermore, everything is made and implemented into a reliable and clear website, made on the PHP framework Laravel.

# References

Picture of globe on cover: clipartfest.com

All chart renders use part of the OpenStreetMap and are rendered in QGIS, as seen on pages 3 and 6, © OpenStreetMap