University of Ghent

Advanced Multimedia Applications

Progress Report

Data visualizations for the curious

A webapplication that allows the user to generically generate visualizations

Bruno Chevalier
Karel Serruys
Pieter Stroobant
Gilles Vandewiele

2014 - 2015

Contents

Part I
Project information

1 Project description and scope

Visualizations usually only show up at the end of a data analysis task. Such a task can be very challenging and a good visualization greatly clarifies the end results. While in traditional data visualization the end user only sees the patterns discovered by the data analyst, interactive data visualization allows the user to explore the data and experiment with new ways of visualizing it.

In the context of a chapter about data visualization in the new course 'Big Data Science', students will get the ability to query and interactively visualize a dataset of traffic and weather data.

Our goal is to design a webapplication, with at least four different visualizations. The user should be able to filter or query these visualizations in all possible ways he could think of. Additionally, the webapplication should support new, but similar datasets and easy creation of new visualizations.

2 Project progress

2.1 First phase (13/02/2015 - 26/02/1015)

Our team was assigned to the first project, visualization of big data. After the introduction lesson, we sat together to discuss a few practical points. Gilles was taking the responsibility to be the team leader, and he was going to take care of the architecture and all communication with external parties, such as the client. Pieter was going to set up our Git to share our code, create a Slack channel as communication platform and create a mailing list to send e-mails to the whole group easily.

At 18/02, we had a meeting with our client. He stated that the most important thing was features, genericity and scalability was less important. We decided to spend less attention to the architecture, thus Gilles and Karel were going to take care of a script that would be deployed on server-side to

cleanse the data while Bruno had to do some research on frameworks that we could use. Pieter was going to take care of the progress report for our first review, while Gilles was going to give the presentation.

2.2 Second phase (26/02/2015 - 12/03/1015)

In the first meeting with the client we were told that scalability and a generic application were far from priorities. However, at the first feedback session, we were instructed to make the application as generic as possible. As in practice, the client mostly comes up with new features or changes in the end. Making the application in a generic manner can overcome potential problems that are paired with this. Thus, Pieter & Gilles were given the responsibility to create an architecture and Karel was now fully responsible for the server-side script, which also had to do aggregation and compression on top of cleansing.

Bruno decided to create the webapplication with the Angular framework and started creating a homepage as examplary code for all other teammembers. Pieter & Gilles updated the progress report with the architecture and mock-ups and Pieter presented it on the second review.

2.3 Third phase (12/03/2015 - 26/03/1015)

Up until now, not many code had been written yet. At 13/03, we got a concerning mail from our client, who said we were kind of overengineering it a bit. He wanted to see something tangible quickly. Also, that day, Gilles visited a software architect to ask for some feedback. He told us that the architecture was good, and would certainly work, but that it shouldn't be that complex for this 'small' project. A meeting was held on 16/03, where everyone was given a development task. Karel handed over the server-side script, that already cleansed the data in a generic manner to Pieter, and started working on a map visualization. The development of other visualizations, such as stacked bar and pie charts, where assigned to Bruno. Gilles was going to develop a start screen where the user could select the data he wanted to visualize. Everyone updated the progress report a bit and Bruno presented it on the third review.

2.4 Fourth phase (26/03/2015 - 30/04/1015)

We were noticing that the development of our webpage took longer than expected, since no one had a lot of experience with Javascript and AngularJS. Also, it was almost impossible to really follow our architecture perfectly, since Javascript isn't really object-oriented. The main success scenario was still roughly followed, but we updated our architecture a bit to be more realistic.

In order to take a huge step forward, we decided to organize a "hackathon", where everyone of us came together to work a whole day on our application. A few days later, we payed a visit to our client to show our progress. He came up with a few extra features. We still had a lot of work, while the exams were getting closer, so we decided to organize another hackathon after the fourth review. Hopefully, to take one of the final steps towards finalizing our project. Again, everyone updated the progress report and Karel presented it, so that everyone has presented our progress once.

2.5 Final phase (30/04/2015 - 14/05/2015)

After the hackathon, the map was upgraded a lot and a hard-to-find bug from the calendar was removed. Also, we build the foundations to create our visualizations (pie, line, bar and multiline), which we're created pretty fast. The links between the visualizations and the calendar and map were made, the lay-out was finished and the dataservice was upgraded.

It was then time to create a final presentation and a nice demo to show off our application. Also, the documentation for future developers had to be written.

2.6 Responsibilities

Task	Responsible
architecture	Gilles Vandewiele & Pieter Stroobant
external communication	Gilles Vandewiele
report	everyone
framework set-up support	Bruno Chevalier
server-side data handling	Karel Serruys & Pieter Stroobant
client-side data service	Karel Serruys & Pieter Stroobant
creation of start-screen	Gilles Vandewiele
calendar	Bruno Chevalier & Gilles Vandewiele
map visualization + timebar	Karel Serruys
piechart visualization	Bruno Chevalier
multiline visualization	Karel Serruys
linechart	Gilles Vandewiele
bar chart	Karel Serruys

2.7 (Non-)Functional Requirements

2.7.1 Must-haves

- A generic script to clean, enrich, aggregate and compress json-files
- Interactive data selection and querying
- At least 4 different interactive visualizations for selected data

2.7.2 Nice-to-haves

- Automatic anomaly explanation: Automatic extraction of web information for explaining uncommon patterns
- Support for real time data addition (and prediction?)

2.7.3 Quality attributes

Usability: Probably used in only one practicum, the learning curve for the students should be as minimal as possible.

Modifiability: In the future, it should be possible to either add new visualizations or apply the visualizations (or at least a part of) to a new, but similar dataset, with a minimum amount of work. As an example, instead of traffic jam data of the Belgian roads, data of routers with their corresponding congestion could be visualized.

Responsiveness: To provide an optimal interactive experience, maximal response times should be low.

Scalability: During the practicum, the number of simultaneous users could be potentially high.

2.8 APIs & Frameworks

2.8.1 Data cleansing & compression

Library: Python 3
Website: `https://www.python.org/`
Description: Programming language
Why chosen?: Python is a high level programming language, with excellent support for importing and exporting the data json files. Furthermore, Python has geocoding modules that allow to enrich the dataset with location info, which is necessary for creating a map visualization.

2.8.2 Developing frameworks & libraries

Library: AngularJS
Website: `www.angularjs.org`
Description: Webapplication development framework
Why chosen?: This framework already incorporated Model-View-Controller, a pattern we wanted to use in our architecture. One of our teammembers already had some experience with this framework. Other possibil-

ities were KnockoutJS [8] and Backbone [2], but no one had experience with these.

Library: Yeoman
Website: `http://yeoman.io/`
Description: Scaffolding tool for modern webapps
Why chosen?: Yeoman comes with generators to quickly set up a webapp that incorporates AngularJS. It was chosen to help speed up the development. We only have to define the frameworks we want to use and the whole scaffold gets downloaded and built up for us.
Yeoman also helps with development through the use of angular generators. If we want to define a new directive for example, we can simply do that using a generator and the new file is automatically generated for us on the correct location including a general unit test.

Library: Compass
Website: `http://compass-style.org/`
Description: Compass is an open-source CSS Authoring Framework
Why chosen?: Compass is the tool of choice when you plan on writing SCSS in combination with Yeoman. Advantages of SCSS over regular CSS can be found on `http://sass-lang.com/guide`

Library: Bootstrap
Website: `http://getbootstrap.com/`
Description: Nice-looking CSS style sheets
Why chosen?: Bootstrap is an open-source initiative that provides CSS style sheets with lots of easy to use classes. We decided to use this to reduce the development time significantly, because styling a webpage can be very cumbersome.

2.8.3 Data visualization

Library: D3
Website: `www.d3js.org`
Description: JavaScript library for visualizing data
Why chosen?: The library was proposed by the client, as he was already learning it himself. Also, D3 offers a lot of different visualizations. One

disadvantage is that a quite steep learning curve is associated with it.

Library: Leaflet
Website: `www.leafletjs.com`
Description: JavaScript library for interactive maps
Why chosen?: The library was again proposed by the client, to use for visualizing the data on a map. There are plenty of similar libraries available, but sources tell this is a very good library as well [1].

Library: Chart.js
Website: `http://www.chartjs.org/`
Description: JavaScript library for simple, clean and engaging charts
Why chosen?: Chart.js creates responsive chart by default. It also gives support for older browser and works well on smartphones and tablets [9].

2.9 Risk list

- The size of the data could be too big at the lowest aggregation level (e.g. frequency of 15 minutes) in order to be processed by our web application. This could be mitigated either be done by only sending data of higher aggregation (short-term solution). Or by implementing a feature that allows to dynamically decide which data the user needs (long-term solution). To minimize this risk, we compressed the dataset in various ways. *We managed to get the size of the dataset to less than 5 MB. Since the context of our application is a practicum where everyone is using a desktop computer, we consider this risk as mitigated.*

- The architecture of our webapplication could not be generic enough to support new datasets. In order to make sure this will not be the case at deliver time, we need to measure how long it takes to implement new datasets which are similar but different, to our given datasets in every development iteration, starting early in the development process. To see how long it takes to make the application work for these datasets. When this takes too long, the architecture needs to be modified (continuous process). *Our application was designed in such a way that it supports every dataset that contains (time, location, value) tuples (value*

occured at time on location) where location is optional.

- Our architecture could also lack genericity so that it becomes cumbersome to implement a new visualization. This will be tested by structuring the development process in such a way that the implementation of the visualizations is done in the end. In other words, the processing of data at server and client-side will be done first, then the visualizations of the processed data at client-side are implemented. This way, we can experience how hard it is to implement our proposed vizualizations. When we feel it takes too long or when the code seems too static, we can update the architecture (continuous process). *We added a start screen that allows users to add their own dataset in the web application. To enable them to get their dataset in the right input format, we developed a well-documented python library. We created a data service to allow developers to create a visualization by creating a new visualization directive or by adding an open source directive. Data for the visualization can be queried by using only a few methods from the data service.*

3 Meetings

3.1 Teammeeting at Technicum (13/02/2015)

Present: Bruno Chevalier, Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Getting acquainted with the other team members, exchanging communication information

Results: The communication information was exchanged and Gilles was appointed to contact the client and create the software architecture.

3.2 Meeting with Dieter De Witte at Zuiderpoort (18/02/2015)

Present: Bruno Chevalier, Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Learning the requirements and the nice-to-haves. Limiting the scope

and scheduling further appointments

Results: The context in which the application will be used was explained and some visualization idea's where shown. It was made clear that we could focus on the collected traffic dataset (or similar datasets), which greatly simplifies the architecture. The client discussed existing libraries and showed several introductory sites to these libraries.

3.3 First review (26/02/2015)

Present: Bruno Chevalier, Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Getting feedback on our first work iteration

Results: We were instructed to make the application generic, nevertheless what we were told by the client. This would make it easier for adding new datasets or visualizations for other people after our project.

3.4 Short standup after first review (26/02/2015)

Present: Bruno Chevalier, Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Dividing work for the following week, taking in account the feedback we just received

Results: We decided there was a great need for an architecture, in order to make our application generic. We want this architecture near to ready by the next review, in order to start implementing early. Also, the data cleansing, aggregation and compression script we wrote, had to be converted to a generic script.

3.5 Second meeting with client at Zuiderpoort (03/03/2015)

Present: Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Inform the client of our progress

Results: The client emphasized that his part of the story is about the visualizations. By the end of the week, he wanted a list of visualizations that we were going to implement.

3.6 Architectural meeting (10/03/2015)

Present: Gilles Vandewiele and Pieter Stroobant

Goal: Create a first version of the project architecture

Results: First architectural UML diagrams, consensus about the architecture.

3.7 Second review (12/03/2015)

Present: Bruno Chevalier, Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Getting feedback on our work.

Results: The professors advised us to use coordinates to map everything generically. Also, the genericity of our design should try to reach the sky, while our real application doesn't have to fit the design completely.

3.8 Short standup after second review (12/03/2015)

Present: Bruno Chevalier, Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Shortly processing the feedback and re-dividing the work

Results: Pieter was going to work on the data-script. Gilles was going to do a little bit more architectural work. Karel and Bruno started learning D3, Leaflet and Angular.

3.9 Meeting with software architect (13/03/2015)

Present: Gilles Vandewiele

Goal: Get feedback on our architecture from an expert.

Results: The architecture was quite okay, he did advise us to replace the pipes & filters-sequence by just one general model, which would simplify our architecture greatly.

3.10 "Crisis" meeting (17/03/2015)

Present: Bruno Chevalier, Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Restructure development process

Results: On 16th of March, Gilles had received quite an angry mail from our client, after showing him the architecture and telling him how the development process was going to be structured. He told us we were "over-engineering" the application. He wanted to see stuff as soon as possible. We decided to start developing the application in parallel with the architecture, taking in to account the feedback we got from Jelle (the software architect).

3.11 Third Review (26/03/2015)

Present: Bruno Chevalier, Gilles Vandewiele, Pieter Stroobant

Goal: Feedback

Results: We gave a demo showing the start screen and visualizations, which weren't linked together yet. The professors told us that we were doing a good job, but still a lot of work had to be done.

3.12 Hackathon I (09/04/2015)

Present: Bruno Chevalier, Karel Serruys, Gilles Vandewiele, Pieter Stroobant

Goal: Make a huge step forward in the development progress

Results: Everyone of us came together to work a whole day on our application. At the end of the day, the server-side script was fully finished. The progress of our webapplication was less tangible, but foundations were placed for further development.

3.13 Client demo 1 (17/04/2015)

Present: Karel Serruys, Gilles Vandewiele

Goal: Show our progress to our client

Results: The client was satisfied: "Good job guys, the foundations are really there, now you just have to build some fancy towers upon them so tourists will come". What he literally meant, was that he was satisfied with our back-end and hoped to see some nice visualizations in the near future.

3.14 Hackathon II (13/04/2015)

Present: Karel Serruys, Gilles Vandewiele, Bruno Chevalier

Goal: Make a huge step forward in the development progress

Results: At the end of the day, the bug in our calendar view was partially fixed. The start screen was fully finished and a track bar was added to the map visualization.

3.15 Client demo 2 (11/05/2015)

Present: Karel Serruys, Gilles Vandewiele, Bruno Chevalier & Pieter Stroobant

Goal: Make a huge step forward in the development progress

Results: The client was impressed and excited to continue working on it later. He also promised us to come look at our presentation.

3.16 Hackathon III (11/05/2015)

Present: Karel Serruys, Gilles Vandewiele, Bruno Chevalier & Pieter Stroobant

Goal: Make a huge step forward in the development progress

Results: At the end of the day, our application, our final presentation and document were as good as finished.

Part II
The application

4 Architecture

The UML class diagram with a high level overview of how our application will look like at startup can be seen in Picture 1. In the diagram, we can see two main components. The server and the client. On the server, data will be cleansed and aggregated offline. When a client wants to start our application through his browser, a json file with information about all files on the server is sent from server to client. The client can then select which datasets he wants to visualize and map the columns of the dataset to the parameters needed for the visualizations. An estimate of how much data will be needed (in KBs) is given (Figure 3). The user also has the option to add and remove multiple datasets and to import his own file. When the user presses the finish button, the Data Service is informed where the required data can be found. The Data Service then starts downloading this data. As soon as the required data is downloaded from the server, decompression, aggregation and grouping is performed. The visualization services are started that use the data from the Data Service and when they are done rendering, they are shown to the user. (Figure 5 & Figure 6).

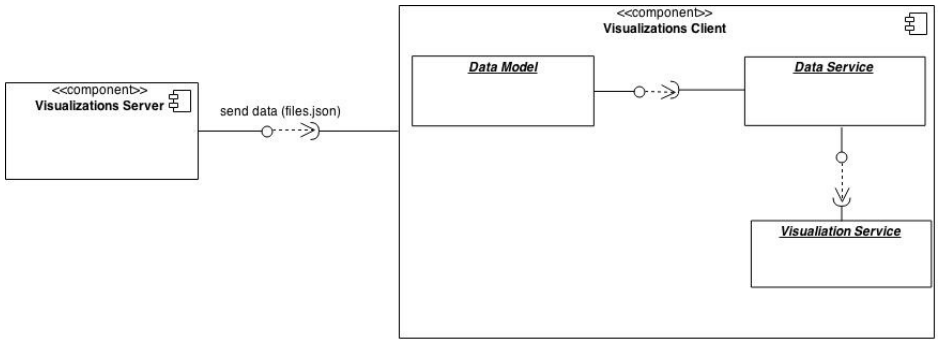

Figure 1: Class diagram

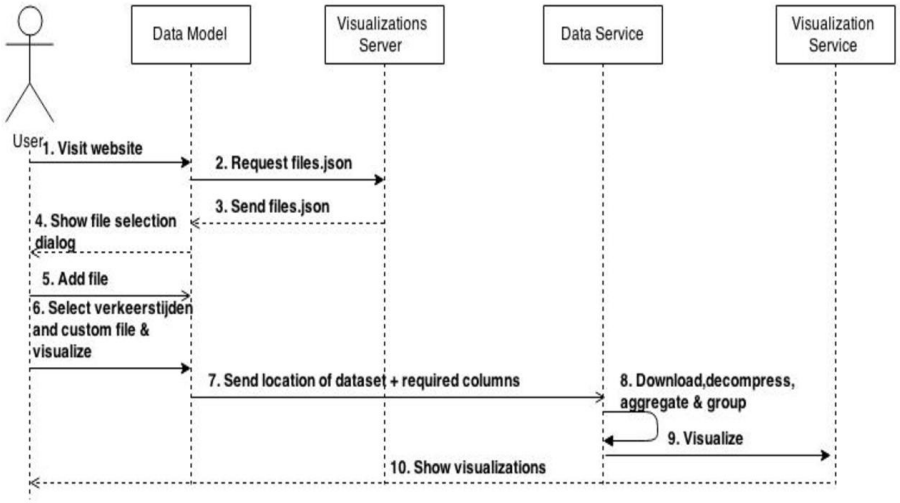

Figure 2: Main flow of the application. User goes to the website, selects the data he wants to visualize and optionally, aggregation and grouping functions can be applied. The datasets' locations are then passed on to the Data Service, where the data is downloaded and then decompressed, grouped and aggregated. The Visualization Services then uses the data in this service to re-render.

5 Client-side

5.1 Start screen

A start screen had to be developed in which the user can select a dataset. When a dataset has been selected, the user has to specify the columns where dates, values and locations can be found.

If the user wants to see some aggregated results, he has to select an aggregation parameter. Options are 'MEAN', 'MAX', 'MIN', 'SUM' and 'COUNT'. 'COUNT' should be selected when the values are actually strings. The most occurring string will then be the aggregated value. Note that up until now, none of the visualizations supports 'COUNT'. See section on future improvements. The other aggregation parameters are self-explanatory.

If the user wants to see grouped data, he has to select a grouping parameter. Here option are 'WEEKDAY', 'WEEKS', 'MONTHS', 'YEARS'. When for example 'WEEKDAY' is selected, the data of all Mondays will be grouped together, the data of all Tuesdays will be grouped together and so on.

A mockup for this start screen was made and is depicted in figure 3. The final result can be seen in figure 4.

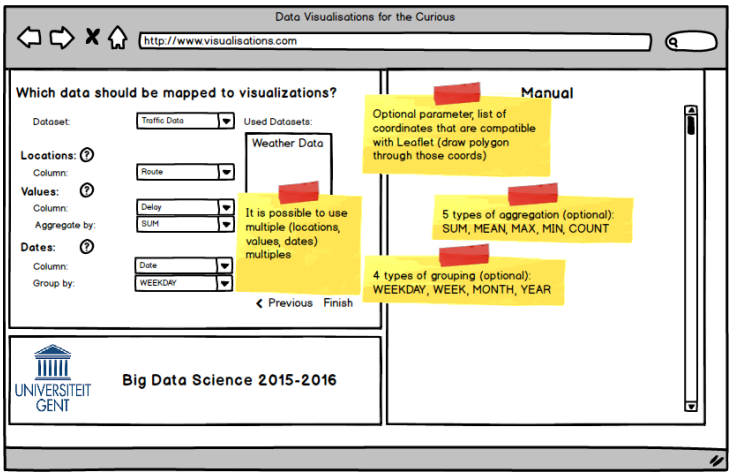

Figure 3: Mock up of the start screen for the client

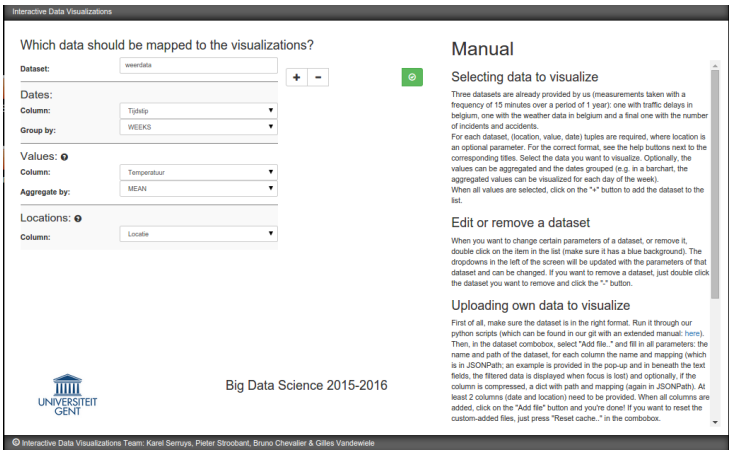

Figure 4: Screenshot of the start screen for the client

5.2 Proposed visualizations

The mockups in figure 5 and 6 already suggest the visualizations we want to create. To give a better overview, they are also listed below.

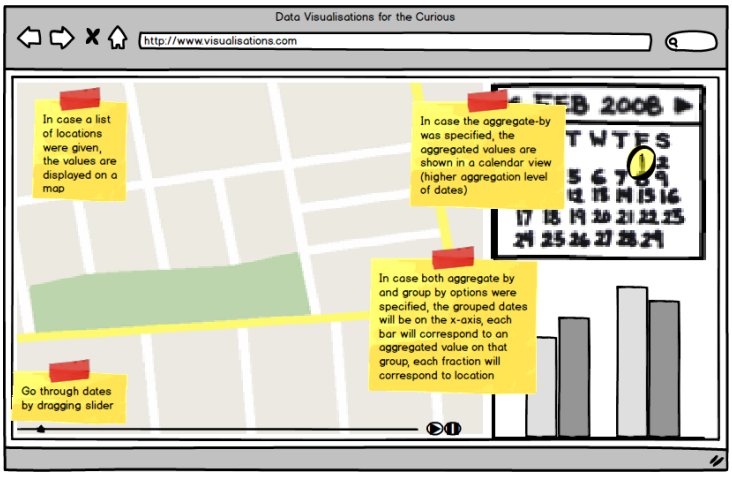

Figure 5: Main application screen

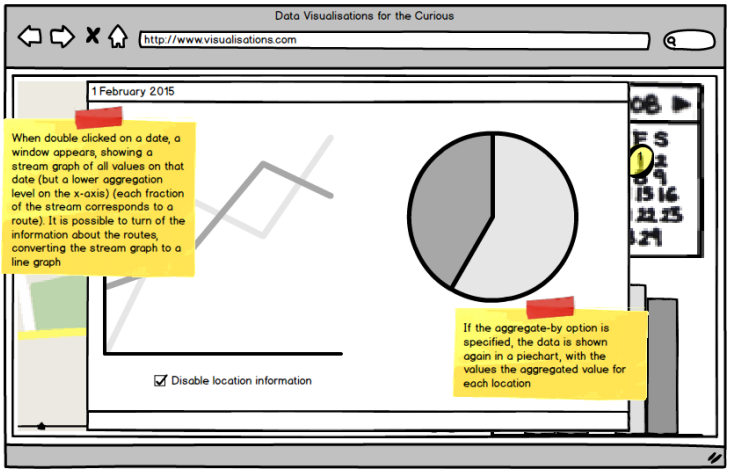

Figure 6: Main application screen

- First of all, we are going to show the traffic jams on a **map of Belgium** (using leaflet) by highlighting the routes where high congestion

is at a certain time (in the form of a timelapse). The current weather will also be displayed at that time.

- A second one is a visualization of the average traffic congestion in specific weather conditions. For every possible weather condition, a **stacked bar** [6] will be displayed. Every fraction of a bar corresponds to a specific route.

- Thirdly, the average traffic congestion will be displayed over time (range of a week) by using a **stream graph** [7]. Again, every fraction of the graph will correspond to a route. As there are many routes, this could become noisy. Therefore, an option will be provided that turns off the route information. The graph will then be converted into a **line chart** [4]

- **Pie charts** [5], with the average weather condition, traffic congestion per road and per weather type will be made.

- Finally, **a calendar view** [3], with the average congestion on a specific day, to see what days in the year had the most traffic jams. Also, the data about the number of incidents, etc. can be shown here (e.g. there are more speed control cameras during the Christmas period).

5.3 Implemented visualizations

The implemented visualizations are listed below and depicted in figures 7,8,xxx and 9. The reader will notice links between the different visualizations were established to ensure a more interactive user-experience.

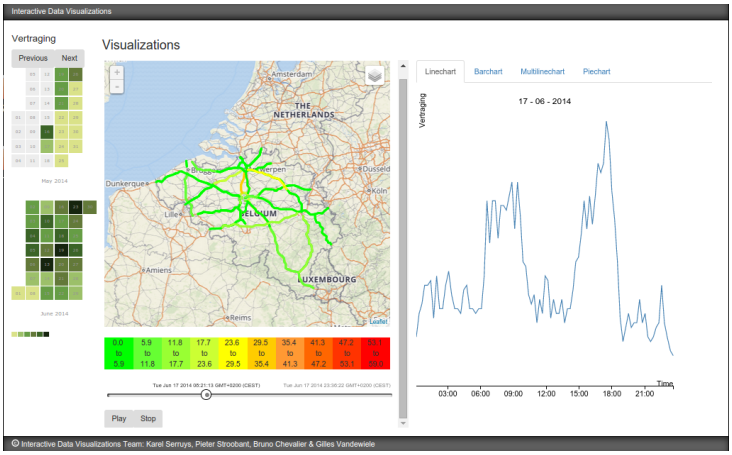

Figure 7: Screenshot of the linechart visualization

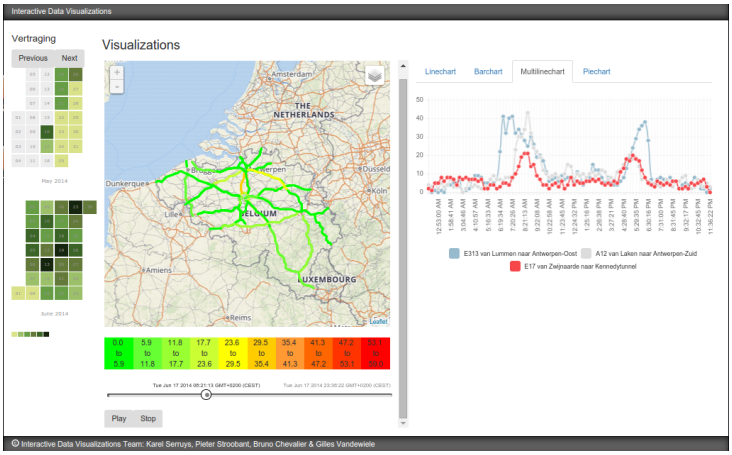

Figure 8: Screenshot of the multilinechart visualization

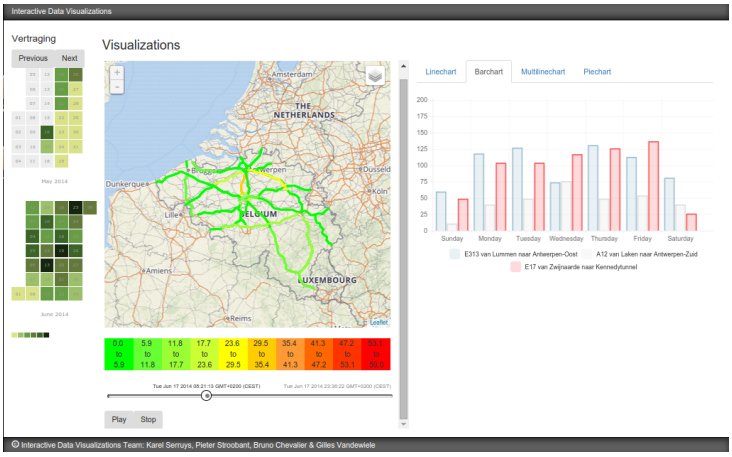

Figure 9: Screenshot of the barchart visualization

- A **calendar view** has been made. This visualization is the basis for the other visualizations that are based on one day. Using a heatmap the aggregation of all values of all locations on an entire day is visualized. When clicking on a day in the calendar, all day-based visualizations will re-render for the new selected day

- Second a **map** was implemented. This map will only be displayed when the selected dataset has locations. The map is day-based visualization. It visualizes all values of every location for a specific time on the map. A timebar can be used to scroll through the a day. Again a heatmap is used to give a notion of the values depicted on the map.

- Another day-based visualization is the **linechart**. This chart visualization the aggregation of the values of all locations for a specific time of the selected day. By hovering over the chart, the values are shown.

- The **multilinechart** is also day-based. By clicking on a specific location in the map, the values of the location on every time instance over the selected day will be visualized in the multilinechart. The chart supports up the 5 locations to prevent it from getting messy.

- The final day-based chart is the **piechart**. This chart .... depends on what Bruno makes ....

- Finally, the last visualization is the **barchart**. This chart will only appear when the user has selected a grouping for the times of the dataset. By clicking a location on the map, the aggregation of all values of the clicked location in a time-group is shown.

All but the second and part of the first proposed visualization were implemented. The reason here is that multiple datasets are not yet supported. See section future improvements.

6 Server-side

To allow the client to choose how the data should be processed and visualized in a way as flexible as possible, much of the functionality of our app is at the client side. The server simply provides the datasets for the client side. However, the data sets we received are hard to process, since they were captured in an unclean and inconsistent way. Furthermore, the datasets were over 250 MB in total. Downloading such large amounts of data would put a significant amount on the server and is also an unpleasant delay for the client. We have created a library of python scripts that allows to clean and compress datasets that are similar to ours easily. Table 1 lists the sizes after each step in the server-side data handling. At the beginning there is one file for every data capture. In the first step each file is converted to a .json file. Then all files are clean. Afterwards these files are put together into one single file. Then the names of the fields are replaced by numbers by using dicts. After that arrays are used instead of objects and finally gzip compression is applied. 'Tot' gives the total size of the data, dicts included.

6.1 Data cleansing

We have created scripts to resolve the most common problems in our data set:

- The captured data were full HTML responses. We are only interested in the body of the HTML messages with an OK response code.
- Non-ASCII characters were captured in a non-deterministic way, but never as the character they really where. We created a script that

Table 1: Dataset size after each step in the server-side data handling.

Step	Size
Start	250 MB
JSON	248 MB
Clean	140 MB
Join	146 MB
Dicts	88 MB
Arrays	11.6 MB
Gzip	2.5 MB
Tot	2.7 MB

recognizes similar words and allows you to correct them.

- The captured timestamps were in an uncommon time format used by the Kimono screenscraper. We converted this to the more well known ISO8601.

The last step of the data cleansing requires converting each datasample to a standard format. Since this is very different for each dataset, this is step requires writing some code. However, this task is usually easy (40-80 lines of code for the different datsets we got) and we have provided both an extensive readme and several example scripts on how to do this.

6.2 Data compression

After the data is cleaned, it can be aggregated and compressed. Since the non-numerical values in a dataset typically have the same (long) name, a lot of compression is possible by creating a dictionary and replacing the value by a reference to that dictionary. We've created scripts that allow to do this in only a couple of commands.

As a last step to increase the compression ratio, we gzipped the data, since gzip is by default supported by most browsers for over ten years. We managed to reduce our 250MB input data set to 3.3 MB.

6.3 Data enrichment

Since creating map visualizations requires coordinates, we have also created scripts that allow retrieving location data based on the location names (this is called geocoding). This geocoding requires the user to verify the search result for each of the searched locations, since errors in geocoding are not uncommon due to unambiguities in place names. If the result was incorrect, the user can enter another query and retry until the results are satisfying. We have also added support to easily retrieve route coordinates based on the start and stop locations.

6.4 Documentation

Since the whole data cleansing process is quite technical and the structure of our library is non obvious, we have created documentation that explains both how to use the library and some of the underlying used mechanisms.

7 Future Improvements

- Multiple datasets should be supported in the visualizations view. This could be done by using a dropdown or buttons to switch between the datasets easily. What would be even better is the possibility to combine two datasets in one visualization.
- The dicts and data could be gzipped, which is by default supported by the most-common browsers. This would ensure even higher compression rates.
- Due to lack of time, we could not realize the count aggregation. This shouldn't be hard to add.
- Our datasets currently got two restrictions. First of all, the frequency of the data should be higher than on a day basis (such as every 15 minutes). Secondly, for each time the data has been measured, the number of locations must be fixed (e.g. 9 weather stations every measurement). Two serious improvements, but hard to realise, would be removing these two restrictions.
- Another improvement that is hard to realise, is dynamic discovery of which visualizations would be useful and with which parameters .
- More visualizations could be supported, we made the architecture in such a way that it becomes very easy to add new visualizations.

References

[1] 50 JavaScript Libraries and Plugins for Maps. http://techslides.com/50-javascript-libraries-and-plugins-for-maps, 2015. [Online; accessed 11-03-2015].

[2] BackboneJS. http://backbonejs.org/, 2015. [Online; accessed 11-03-2015].

[3] D3 Calendar View. http://bl.ocks.org/mbostock/4063318, 2015. [Online; accessed 11-03-2015].

[4] D3 Line Chart. http://bl.ocks.org/mbostock/3883245, 2015. [Online; accessed 11-03-2015].

[5] D3 Pie Chart. http://bl.ocks.org/mbostock/3887235, 2015. [Online; accessed 11-03-2015].

[6] D3 Stacked Bar Chart. http://bl.ocks.org/mbostock/3886208, 2015. [Online; accessed 11-03-2015].

[7] D3 Stream Graph. http://bl.ocks.org/mbostock/4060954, 2015. [Online; accessed 11-03-2015].

[8] KnockoutJS. http://knockoutjs.com/, 2015. [Online; accessed 11-03-2015].

[9] The 15 best JavaScript Charting Libraries. http://www.sitepoint.com/15-best-javascript-charting-libraries/, 2015. [Online; accessed 5-05-2015].