University of Zurich[UZH]

Department of Informatics

DEVELOPMENT OF ADAPTIVE HEATMAPS FOR INTERACTIVE FEED EXPLORATIONS

Bachelor thesis of

Dominique Christina Hässig

Matriculation number: 10-933-257

Email: dominiquechristina.haessig@uzh.ch

Supervisor: Prof. Michael Böhlen

Date of submission: June 17, 2020

ABSTRACT

Deutsch

Das Ziel dieser Arbeit war die Weiterentwicklung der Visualisierung der Schweizer Futtermitteldaten.

Es gibt verschiedene Möglichkeiten, die geographische Verteilung von Messwerten auf einer Karte darzustellen. Eine Möglichkeit stellen Heatmaps dar. Wichtige Faktoren für eine nützliche Darstellung sind dabei der Radius und der Gradient der benutzen Farben. Ein Hauptziel dieser Arbeit war die Erstellung korrekter Heatmaps in beliebigem Zoomlevel und mit beliebig vielen Daten.

Ein weiteres Ziel der Arbeit war die Migration der Schweizer Futtermitteldatenbank vom älteren Webapplikationsframework AngularJS zum neueren Angular. Dabei wird erklärt, wie der Aufbau der Seite nun in der neuen Version aussieht und funktioniert.

English

The goal of this thesis was the further development of the visualization of the Swiss Feed Database.

There are multiple ways to present geographical distribution of data on a map. One way poses heatmaps. Major parameters for a useful presentation are the radius and the gradient of the used colours. A main goal of this thesis was the construction of correct heatmaps in arbitrary zoom level and with arbitrary many data.

Another goal of this thesis was the migration of the Swiss Feed Database from the older web application framework AngularJS to the newer Angular. In doing so it is explained, how the structure of the website now appears in the new version and how it works.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to Prof. Michael Böhlen, who not only gave me the opportunity to work on an interesting and challenging project, but also has guided me through the entire development of this thesis. Although our personal meetings were rare due to Covid-19 pandemic, the Skype meeting were always of great help.

I also want to thank Annelies Bracher from Agroscope, who helped with many useful feedbacks from the user side regarding the functionalities.

A special thank goes to my brother Nicolas. Since he has a lot more experience in working with css files, he helped me a lot with the revising and improving them. Likewise, the whole Sunday at his work for the steady internet connection for the measuring was very useful.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Topics of the bachelor thesis

The first topic of this bachelor thesis was the migration of the Swiss feed database homepage from the web application framework AngularJS to its newer version Angular. AngularJS was the first version of Angular, which was rewritten in many parts for further versions because of performance and usability reasons. The whole architecture has changed a lot in this migration. Therefore, the description of the new architecture was also necessary, which is part of this thesis.

In the next step, the existing functions had to be improved and new functionalities had to be added. One respective goal was the improvement of the stability and usability of the homepage.

The heatmap for example, had problems with different erroneous overlays, which had to be fixed. Further, a new functionality is introduced for changing the colour gradient or the radius of the heatmap for giving the user the possibility to do a more differentiated data analysis.

Many of these improved and new functions, which belong to one component, need to change other components than itself as well. For that, a way had to be found for a fast and correct data flow. Therefore, this data flow was changed to comply with this goal. In this new data flow management, the number of requests was minimized and optimized. For example, some requests were parallelized. The new data flow is described here.

The last part of the thesis was the evaluation of the implemented solutions. How did they improve the usability? What is the impact of the coding changes regarding speed and size? How does the implemented solution scale for arbitrary many data points? How much time need the subparts of a visualization process? With various measurements, these questions should be answered.

1.2 Overview

In chapter 2, the non-technical background of this thesis is presented. Here, it is explained what the general environment of the work. What is Agroscope, for which the thesis was made? What is the feedbase website?

In chapter 3, the technical basis is provided. What is the used framework, which programming language was used and in what IDE (integrated development environment) was it developed?

In chapter 4, the structure of the feedbase website is explained. This structure is majoritarian new, because a big part of my work was the migration of the old version to this. Further, the new and improved functions are presented. Here, I tried to explain

especially, what I have done for the improvement. Lastly, I present with the help of two use cases, how all parts of the architecture work together.

In chapter 5, diverse measurements of the new version are presented and explained. What are speed improvements? Where and why do we get long loading times? Question like these have been tried to be answered in this chapter.

In chapter 6, I will discuss the measurements and try to generalize answers to the questions. Further, I will show possible topics for future work related to the feedbase homepage.

In the appendix, the detailed tables of the measurements done in chapter 5 are shown.

2 BACKGROUND

2.1 Agroscope

Agroscope is the Swiss centre of excellence for agricultural research. It is an associate of the federal office for agriculture (FOAG). The research includes the entire value chain of the agriculture and the food sector. The aim of Agroscope is a competitional and multifunctional agriculture, high-standard food for a healthy nutrition and an intact environment. (Agroscope, 2020)

2.2 Feedbase.ch

Feedbase.ch is the homepage of Swiss Feed Database, which models the concentration of nutrients in feed samples collected across Switzerland. The homepage, which is developed by the UZH in collaboration with Agroscope, is used by Swiss farmers and research institutions.

The first version of the homepage was created back in 2007 in collaboration with the ETHZ. This version replaced for the first time the print version of the feeding recommendation and the nutrient tables.

In 2013, the homepage was developed further to a temporal Data Warehouse, now in collaboration with the Database Technology Group of the university of Zurich led by Prof. M. Böhlen. The new version allowed beside average requests also requests to single values with geographical and temporal information. They also revised the layout and added new visualization functions. (Bracher & Boltshauser, 2013)

The data is collected from three sources:

- research data
- praxis data and surveys
- literature data, data bases

The foundation of the data is generated by the centralization of the nutrient tables of ruminants and pigs, which are continuously expanded by data of current research projects. Further is the data pool enlarged and actualized in partnership with AGRIDEA, feed labs and feed companies. Where no data is available from Swiss test results, it is taken from publications and nutrient tables of foreign institutions like the French National Research Institute for Agriculture, Food and Environment (INRAE), the German Agriculture Society (DLG), the Dutch feed agency (Centraal Veevoederbureau - CVB) and the National Reasearch Center of the USA (NRC).

The Swiss Feed Database uses as database PostgreSQL. The server side is programmed in the web application framework Express.js on the JavaScript-based platform Node.js. For the front-end, the web framework AngularJS was used until this bachelor thesis. Further, the homepage uses Google Maps for geographical presentation.

3 TECHNICAL BASIS

AngularJS was the first version of Angular, which was first released on October 21st 2010 (Angular, 2010). Because of many shortcomings of the first version regarding performance and usability, the development team spent a year rewriting the code from scratch. The new version was then released in late 2016, renamed to Angular for version 2 and following versions. Since then, new versions were released, but none of them had such a drastic change in the coding style as back in 2016. At the present day, version 9 is in use.

3.1 Angular

Angular is an application design framework and development platform developed by Google. It specialized for single-page client applications and uses HTML and TypeScript.

The basic building are NgModules, which provide a compilation context for components. These NgModules can import functionalities of other NgModules and be used by other ones. (Google, 2020)

An Angular app has at least a root module, conventionally named AppModule, which enables bootstrapping. This root module is usually accompanied by many more feature modules.

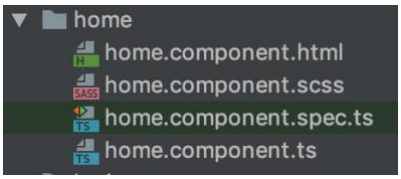


Figure 3.1 - Component elements using the example of home

The previously mentioned components are the visible elements on the screen with the programmed logic and data. The core code of the component is in the basic TypeScript file (in Figure 3.1 *home.component.ts*). They are identified in the code by the *@Component()* decorator. Every component defines a class that contains application data and logic. Further, each is associated by a HTML template (in Figure 3.1 home.component.html), which defines the displayed view, and a related component-specific metadata (in Figure 3.1 *home.component.spec.ts*). The last part, that usually belongs to a component, is a CSS (Cascading Style Sheets) file. There are other style sheet language, which can be compiled in CSS, like SCSS (Sassy CSS), which is used in this project (in Figure 3.1 *home.component.scss*).

Functions in a component are related to its view. For other functions, which are used by multiple components, services are used. Services provide specific functionality not directly related to one view. Example for such functions are data fetching or notation of events between multiple components. Since the services can be injected into components as dependencies, the code can be made modular, reusable and with that efficient.

An app has usually many different views, which are arranged hierarchically. An example for this will be shown in Chapter 4.1, where the architecture of the feedbase homepage will be explained. Although Angular is specialized for single-page application, it is possible to navigate to different views. For this, Angular provides the Router service.

The organization of the code into distinct functional modules helps in managing development of complex applications. It also takes advantage of lazy-loading, meaning only the needed code is loaded at start up and further code on demand.

A very good introduction in the building of an Angular app is provided by Angular itself in the Tutorial: Tour of Heroes. (Google, kein Datum) Here, all important functionalities are explained step by step and with useful code examples, which can be worked through in relatively small time. As the steps between are also shown well, it is easy to roll back to some steps, if one needs so.

Another useful support gives Angular material library. (Google, 2020) It provides many components, that are useful for creating a page. In the documentation are many different examples listed for different use of them, so that it should be easy to customize it for one's own need.

3.2 AngularJS vs. Angular

As mentioned at the beginning of this chapter, AngularJS had problems with performance and how the framework worked. For overcoming these problems, the code was rewritten in big parts. Some of the resulting changes will be discussed consecutively.

AngularJS follows the traditional MVC (Model View Controller) architecture. The model is the abstract presentation of the data, the view represents the presentation layer and the actual user interface, and the controller represents how user interactions are handled and binds both the model and the view. This is a big difference to Angular, which uses as explained in the former section a component-based architecture. (Manjunath, 2018)

The concept of modules also differs drastically between the two. A NgModule, the module of Angular, is a container for defining a functional unit. In AngularJS a module consists of a container for different parts of the application, like controllers, services, filters, directives etc. (Google, 2020)

Regarding HTML, the most important change took place, that every component in Angular has its own template attached to it. In the previous version, AngularJS-specific code was added to a view. Additionally, a lot of new features were added in the newer version.

The JS in AngularJS stands for JavaScript, since it is a pure JavaScript framework. Angular changed the default language to TypeScript. TypeScript and the upsides of it in comparison with JavaScript are explained in the next section.

3.3 TypeScript

Angular is generally written by using TypeScript. This is a typed superset of JavaScript, which will be compiled to JavaScript. Due to this predisposition, this code written in TypeScript can be run on almost any machine that exposes a JavaScript API, independent of its version. (Fenton, 2014)

In contrast to JavaScript TypeScript can perform optional static type checking, which prevents unnoticed compile-time error. This makes the code more predictable. (Manjunath, 2018)

3.4 WebStorm

WebStorm was the used IDE (integrated development environment) in this thesis work, since I already got familiar in a course at the university. It was developed by JetBrains and firstly released in 2010.

WebStorm provides advanced coding assistance for JavaScript and compiled-to-JavaScript languages (like TypeScript). It supports deeply popular web frameworks like React, Vue and, which was important for this project, Angular. Further, it also gives assistance for Node.js, HTML and CSS.

Intelligent code completion and on-the-fly error detection help preventing many errors. Another useful aspect is the easy handling of version control system with WebStorm (as well as other IDE's from JetBrains). Changing between different commits and branches is easily done in the IDE and review changes can be seen without much effort. (JetBrains, 2020)

4 FEEDBASE HOMEPAGE

4.1 General Architecture

First, we look at the general structure of the migrated homepage from AngularJS to Angular. Therefore, we take a detailed look to the component structure and the services, that link the data of the diverse components, followed by a detailed explanation of the data flow between the different main components of the homepage.

4.1.1 Components

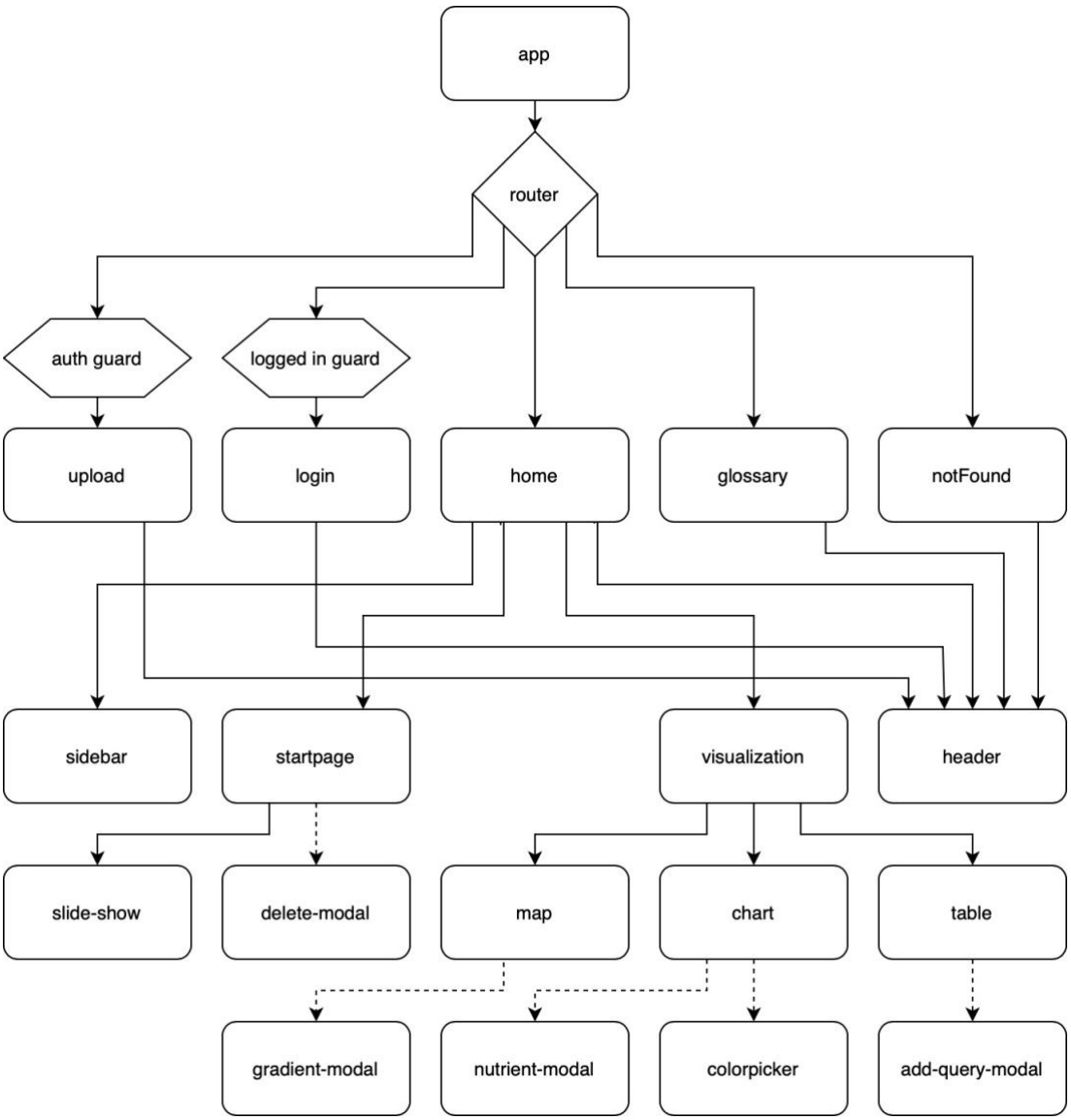


Figure 4.1- Component Structure
Rectangular: components; Hexagon: guard; Rhombus: router;
Solid line: component called in html code of parent component;
Dotted line: component called by function in parent component

As typically for an Angular project, the structure is constructed hierarchically, as it is presented in Figure 4.1.

The app is the basic component. From this point we get to the following pages, depending on the specific corresponding page selected. If an arbitrary subpage is selected, which name does not exist or to which no connection is made in the router, one will get to the “Not found” page. “Home” is the page loaded when no subpage is selected. “Upload” and “Login” have a guard, meaning only users, who satisfy the respective conditions, will be allowed to enter these pages. Otherwise, they will be sent back to the “Home” page. The condition for the “auth guard” is to be logged in with an administrator user. The “logged in guard” checks if the user is already logged in and if so, deny the access to this page to him. This way, one can only be logged in once per browser, which prevents errors from multiple usernames. This is especially important when it comes to the user level (the access level of the specific user). Otherwise, it wouldn’t be clear which username and accordingly which user level is meant for the request. Another solution would have been to overwrite the user when logging in a new user.

Figure 4.2 presents a code of the function “canActivate”, which checks if the user can open the upload page. For this reason, the server is asked (in the serverService), if the user is authorized to enter the page. If the answer is an error with status 401, which is the status for unauthorized requests, the user is navigated back to the start page. Otherwise, the user is allowed to enter the upload page, the function returns true and the page is loaded.

```
canActivate(): Observable<boolean> {
 return this.serverService.upload().pipe(
   map((res: any) => {
    if (res.status === 401) {
     this.router.navigate(['/']);
     return false;
```

Figure 4.2 - Code of the admin guard
The function checks if the user is allowed to enter the page /upload

If one further proceeds in the component structure of Figure 4.1, one will see the header. All pages include the header, which contains menus for opening the side bar, and information related pages. Further, the logo of the homepage is always seen, which leads always to the start page, the login link and the three available language options.

Most of the functions of the feedbase homepage are on the “Home” page. The first view that the user sees is mostly constructed out of the components of “start page” view. This view includes a slide-show component with pictures of different feed types and a “delete-modal” component, that can only be used by the admin and which is used for a modal window to delete a no longer needed predefined query. The content of this view are the predefined queries (called top queries), the news, the project leaders and the partners. The top queries and the news are loaded from the database on opening this page.

The “delete-modal” component, as well as the other modal window component, are not included in the html code of the page and a matter of fact not loaded while loading the component. They are only loaded by their respective functions in the TypeScript code

```
<div>
  <div id='visualization-wrapper' class='row display-flex'
          *ngIf="data.datatype === 'td'">
    <app-map id="map-visualization" class='col-lg-6'></app-map>


    <app-chart id="charts-visualization" class='col-lg-6'></app-chart>
  </div>
  <div id='radius-search-info'  *ngIf='radiusData !== null'>
    <span>{{radiusText}}</span>
```

Figure 4.3 - Html of the visualization component

when needed. All of them will be loaded as smaller overlaid windows over the existing view.

The “side bar” component is used to select individual filter options. It can be opened by the previously mentioned menu in the header. Since one can select and change filtering options in the “start page” view and in a selected “visualization” view, it is situated on the same level as these two views.

From the “start page” view, either by selecting a prestored query on this page or by creating own selecting criteria over the “sidebar”, we get to the “visualization” view. This is the view with most of the functions of the homepage. It consists of the three subcomponents “map”, “chart” and “table”, which each of them presents the data in different ways and provide different functions. For queries with summarized data, the only component in the visualization is the table. The table has an “add-query-modal”, which only can be seen by admin users. This is a modal window, where the current query can be saved as a new query to the top queries at the “start page” view. The chart has two components for modal windows, the “colorpicker” component for changing the colour of the chart and the “nutrient-modal” component for changing the nutrients for the visualization in the chart. The map has also a modal window, which is the component “gradient-modal”. This component is for changing the colour gradient for the overlay.

We can take a deeper look in the html code of the visualization component (Figure 4.3). The upper part of the page is only loaded, when the data type is td, meaning, the data is detail data. This is controlled by *ngIf=”data.datatype === ‘td’. This part consists of the map and chart component. These included components are by name conventions named app-<component name>, e.g. app-map and app-chart. The next part is the radius search part. If the user does a radius search, the centre of the circle and the radius size are displayed as well as a button to clear this radius search. The last part, which is always loaded, is the table component. The table data, the username and the user level, which are already loaded in the home component, are passed down to the table component.

4.1.2 Services

The feedbase homepage uses four own services for functionalities and data of the many components. These services are the language service, the query service, the server service and the sidebar service.

The language service provides functionalities related to the language. The integrated functionalities are the check of the language in the cookies and the saving of language changes. Additionally, it is responsible, with the help of the internalization library of Angular (called translateService) for the correct translation of the words in different languages. The information of the translation is in JSON-files with the name of the respective languages. For the feedbase homepage, the languages are German (de), French (fr) and English (en).

The query service is the core of the data flow. It is responsible for all queries and with that data changes of the different parts. Since the data are not just changed from the respective components, they have to be handled separately from them. By this way, other components can interact with the data, which will take place quite often. The data flow will be discussed in the next section in detail. The components observe their own data in this service and react on changes on them.

The server service covers all requests to the server. Most of the function calls in the server service come from the query service, which handles the returned data, but some simpler requests are also done directly by some components.

The sidebar service handles the component overlapping functionalities belonging to the sidebar. This functionality is achieved by the opening and toggling of the sidebar. Although it is a functionality of the sidebar, it is started by the header component.

4.1.3 Data Flow

Most of the functionalities of this homepage are, as previously mentioned, in the "visualization" view. For many of these functions it is required to load and visualize new data. That's why I will now explain these functions and which respective components are affected by them. The data flow is visualized in Figure 4.4.

With selecting a query on the start page, one loads all visualization parts, if it's a detailed data query and not a reference data query. If it is the second, only table data is loaded. The selection of a query also updates the sidebar. The other way of selecting a query is by filtering in the sidebar.

The map, which is shown on a google map, has three main functions, which have influence of the visualized data. The first is the selecting of the map overlay. The options are:

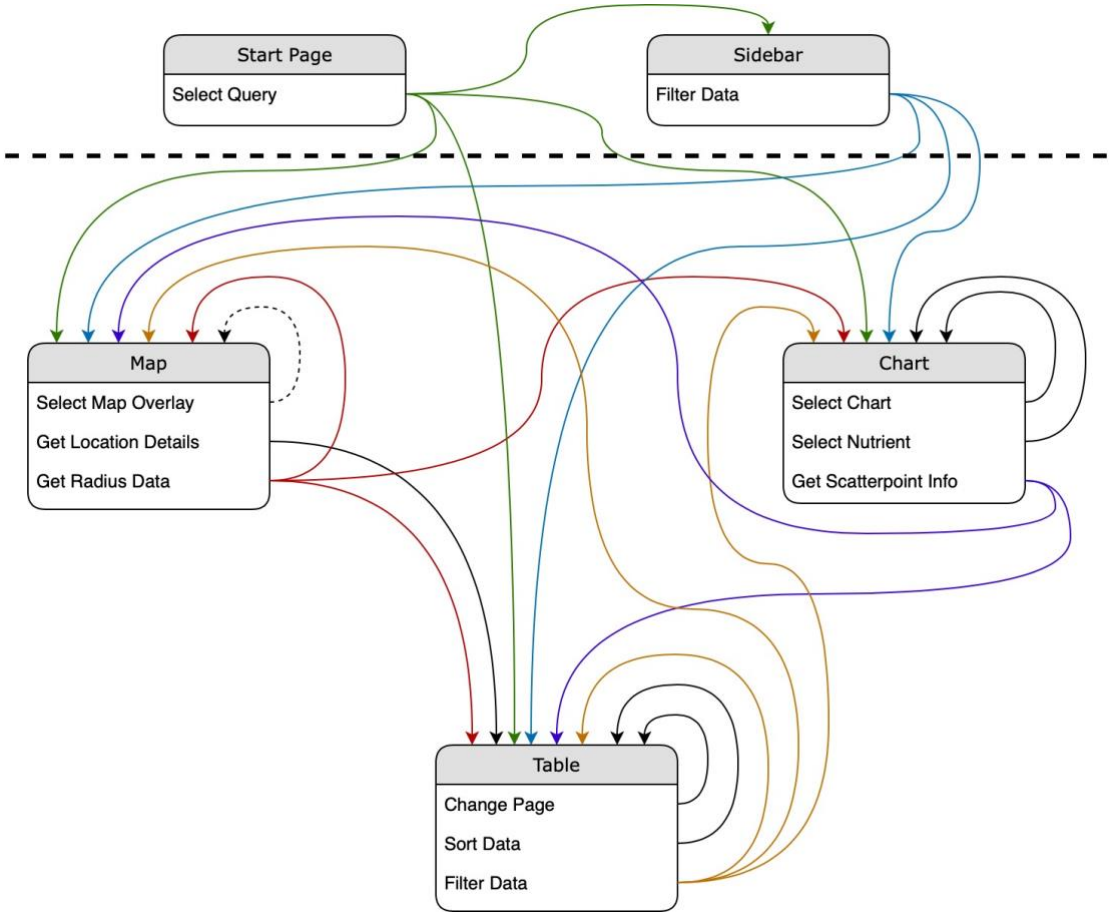


Figure 4.4 - Data Flow between components
Colour is for easier tracking of data flow to multiple components. The components over the dotted line are entry points while the components under it belong to the visualization. A dotted arrow indicates only loading of data at specific times

1. Location: Depending on the zoom level, one can see either the canton view or the location view. In the canton view, one can see the number of data points per canton, visualized by a colour gradient. In the location view, each data point has a marker.
2. Density: This overlay shows the spatial density of all data point belonging to the current query.
3. Regression: The regression shows a heat map corresponding to the values of the selected nutrient. This nutrient can be changed in a submenu of the regression option.
4. No data: By selecting neither of the previous options, one can see the standard google map.

Unlike in other functions, which will be discussed later on, the option choosing does not always require loading new data in this new version of the homepage. As soon as one overlay is loaded, it stays in the cache. This way, the changing of the views can be done very fast, since the longest part of the overlay changing, the loading of the data from the server, has not to be done every time. This will help with the comparing of results.

In the location view of the location option, one can select a point in the map and choose to get the location details. With this intend, the map starts within the table component the order to load in the first rows the data of this location.

In the same location where you can select the location data, one can also select to get radius data. Doing this, a circle with an initial radius of 5 km is loaded around the selected point. This radius can be changed freely. The first loading and every radius change also changes every component of the visualization view. In the map only the data of the view are shown. This is also the case, if one changes the overlay. In the table and chart are only the data of the location presented in this area.

In the chart are also three main functions related to data change. Similar to the map, one can select between different chart types, namely scatter chart, correlation chart, box plot and violin plot. An additional option here is to get a summary table of the data of the query. The change of the chart only affects its own data.

The same is given for the second function, the selection of the nutrient. The user has the possibility in a modal window, to change or add the nutrient which he wants to be shown in the chart.

The last function in the chart is only usable in the scatter chart. By clicking on a respective point, the chart gives the table the order to load the data of this point in the first given rows, similar to the location details in the map. If only one location point belongs to this data point, the map zooms to its location.

The table has also three functions which have impact on the loading of data. One is the changing of table pages. Since only 50 rows are shown at once, new data have to be loaded on every change of the table page.

Another function, which also only affects the table data, is sorting. The user can sort for every column in both directions.

The last main function of the table is a function, which can give many insights in the data. This function is filtering. One can filter the data for specific data ranges. The filtering is not only done in the table, but also in the chart and the map. This way, one can see where the filtered data comes from and how is the behaviour of the filtered data.

There are also other functions, which are not presented in Figure 4.4, since they don't change the data of the respective visualizations. In the table component and in the summary part of the chart component exists the possibility to download an excel file with the data of these tables. If the table data is detailed data, the download is only available for registered users. As mentioned in 4.1, the admin has the possibility to save the current query in the top queries of the start page. Two other functions, which only change the way the data is presented in the map, but without changing the data are the functions for changing the colour gradient and the radius of the heatmap. Although the view changes with these functions, the data itself is not changed and therefore doesn't have to be again loaded.

4.1.4 Example

For illustrating the structure and data flow, I will now explain on the basis of an example what happens in the background of the homepage.

I open the page on start page.

By opening the page, the index.html of the page will be loaded. This loads the component “App”, which itself loads the router. As the page route ends without an additional path, the “Home” component is chosen. As no table data is yet loaded, the component loads the “Startpage” component. Initializing this “Home” component subscribes the language service. This checks if the saved language is the same as the language of the service and request, if this is not the case, the start info comes from the server. This request is sent from the “Home” component instead of the “Startpage” component, because the data of the username and level is also needed in other components, for example the header, which is also loaded by the “home” component. As the local language is not initialized at the beginning and the language service variable is either the saved language of the cookies or German, the start info will always be loaded at the beginning. By filling this loaded information in the html files of the involved components, the start page is presented as seen in Figure 4.5.

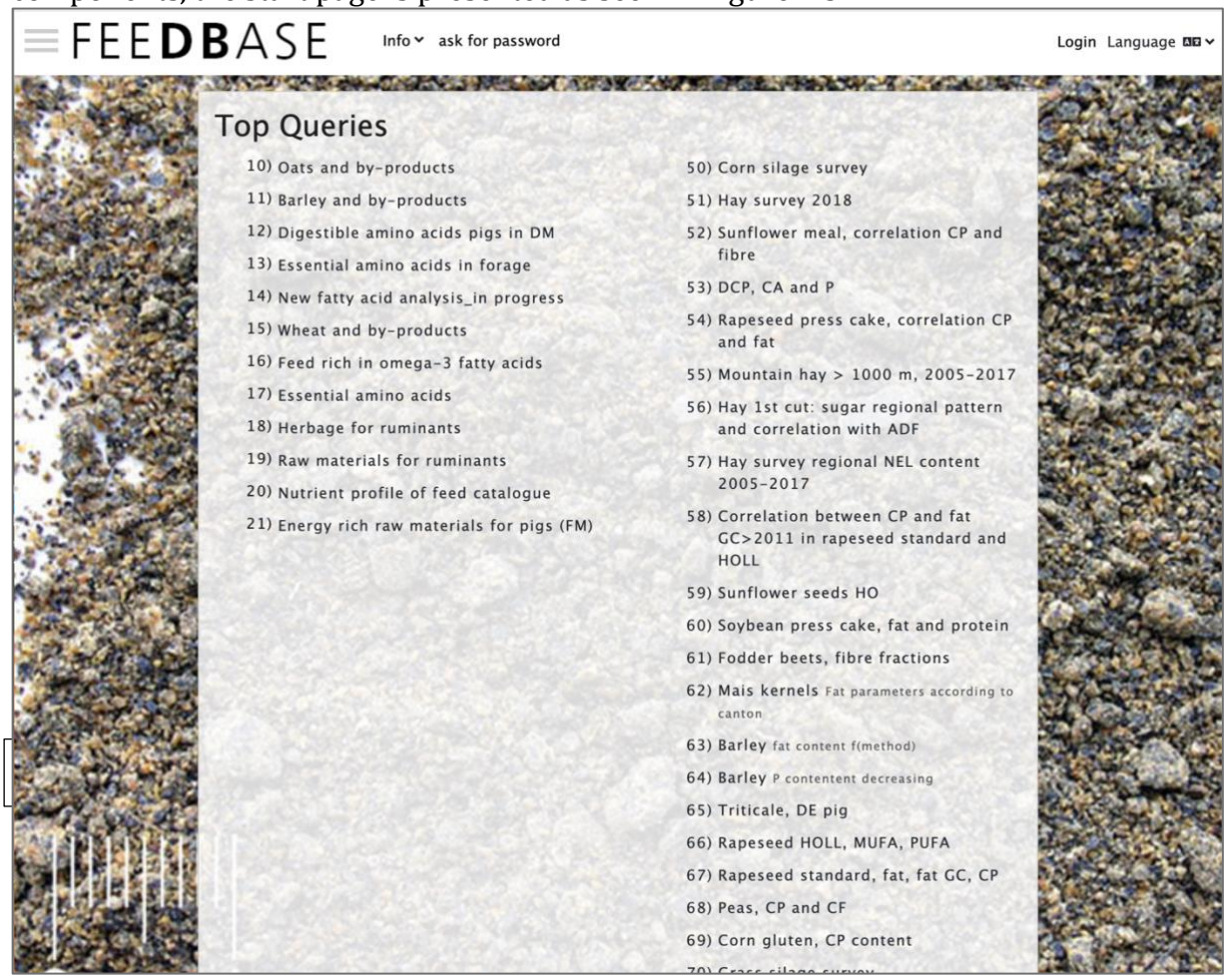


Figure 4.5 - Start page of the Swiss feed database

I select the top query “Hay survey 2018”.

By clicking on the query, the function “selectQuery” is called, which requests the server with the help of the server service for the parameters of the selected query. The query is saved in the database with an id (in this case 898). The received data will then be saved in the corresponding filter parameters of the query service.

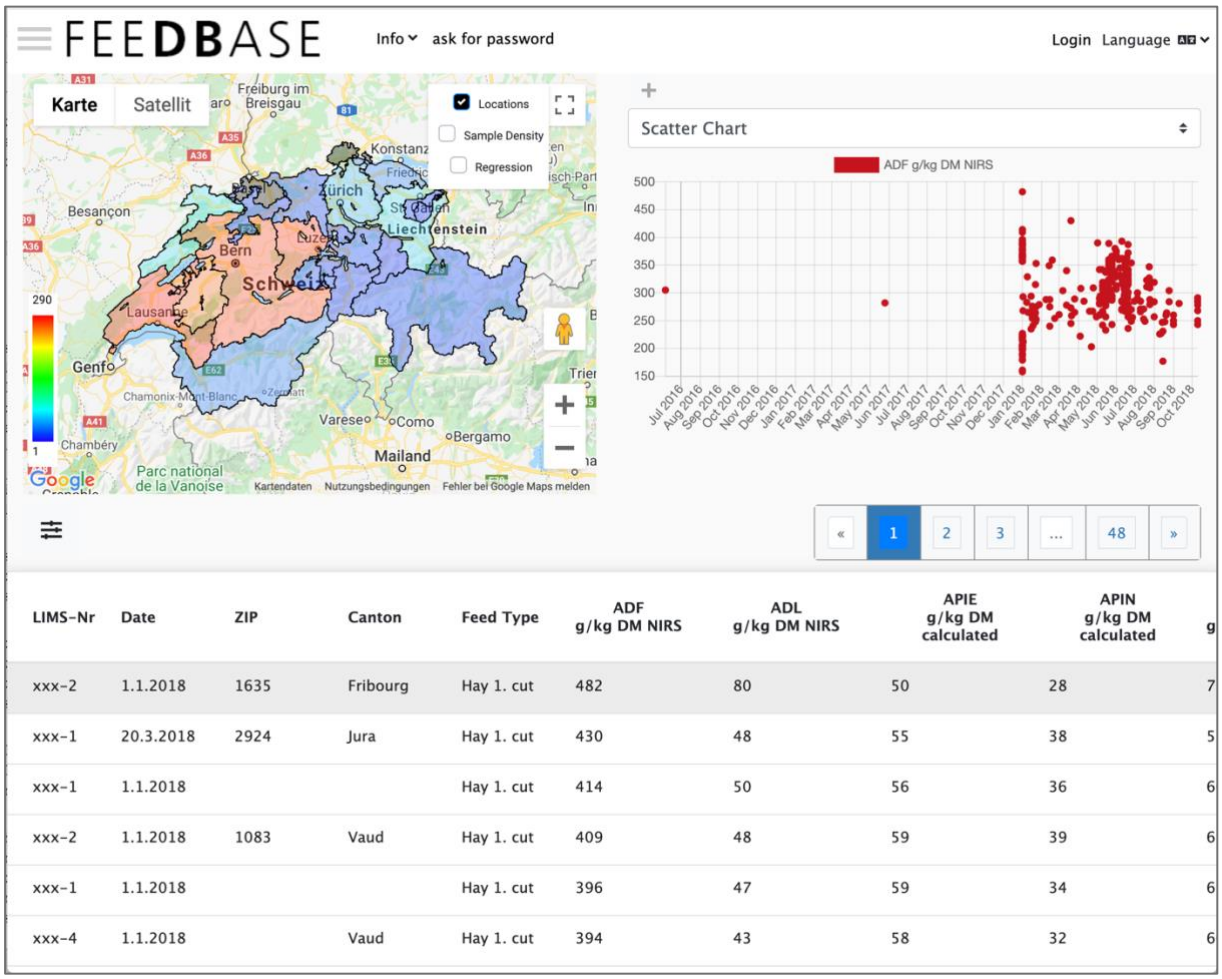
Next, a request for data of visualization parts will be loaded. As the data type of the query is ‘td’, meaning detailed data, the chart and map data will also be requested beside the table data. For both the chart and the map, initial visualization types are defined (for chart scatter chart and for map location data with zoom level 7). These types define which specific data request has to be sent (hull for chart, mapCantons for map).

For this query, the first returning request is the chart data, followed by the map data. These data are saved in the to the data corresponding subjects with type and column data. The column data was already loaded with the parameters in the first sent request. For the map data, the zoom level is also saved.

As soon as the table data has also been returned, the data of it is saved in its subject. The home component has an observer of this object, which reacts to this change. As table data now exists, the “Startpage” component is no longer loaded. Instead, the “Visualization” component is loaded. As the datatype is ‘td’, all subcomponents are loaded, meaning the map, chart and table component. The code of this loading was explained in detail in section 4.1.1 and shown in Figure 4.3.

When loading the table component, the visualization component sends to it the table data, which is observed by it. This starts the function “createTableParams”. If no columns are saved yet in the component, which is the case for the first loading, the displayed column names are first loaded in an array. There are fixed names like the lims number, date, postal code and canton as well as the names of the different nutrients of the query. Next, the row data is saved in the correct data source form for the table. As soon as the data source is created, the table is shown to the user.

The map component subscribes right after the initialization the map data observable. It is subscribed after initialization and not on initialization, because if data exists on the initialization, the map should be created, but for this the component, which is about to be initialized, has already to be initialized. As the data is already existing (since it was returned before the table data), the map with data will be created right away. First, the google map will be created with Switzerland in the centre and zoom level 7. This is followed by the computation of the overlay. As the zoom of the data is 7, which is smaller than 10, and the data type is “showMarkers”, the function draw cantons is called. The map gets a new listener for zoom change, which loads the point data as soon as the zoom level is 10 or higher. The canton data is added with google maps, as well as the shape colour, which is defined by the “quantity normalized” property of the different polygons. Further, two additional listeners are added. On one hand a listener to clicking on a polygon is



LIMS-Nr	Date	ZIP	Canton	Feed Type	ADF g/kg DM NIRS	ADL g/kg DM NIRS	APIE g/kg DM calculated	APIN g/kg DM calculated	g
xxx-2	1.1.2018	1635	Fribourg	Hay 1. cut	482	80	50	28	7
xxx-1	20.3.2018	2924	Jura	Hay 1. cut	430	48	55	38	5
xxx-1	1.1.2018			Hay 1. cut	414	50	56	36	6
xxx-2	1.1.2018	1083	Vaud	Hay 1. cut	409	48	59	39	6
xxx-1	1.1.2018			Hay 1. cut	396	47	59	34	6
xxx-4	1.1.2018		Vaud	Hay 1. cut	394	43	58	32	6

Figure 4.6 - Visualization of "Hay survey 2018"

added, which shows the number of data points in the canton, and a colour legend for the canton colours. On the other hand, a listener to clicking on the colour legend is added, which would open the “gradient-modal” component for changing the colour gradient or the heatmap radius. The colour legend was created with the computation of the overlay.

The chart component subscribes the map data observable like the map component right after the initialization for the same reason. First when setting up the chart, a tag is added. This tag saves the information of the nutrient to be shown, including displayed name and colour of the data points. After that, the actual chart is created. As the type is ‘scatter’, the function “drawScatterChart” is called. First, the data will be processed in the correct form for the dataset of the chart. Next the chart itself is created with all necessary information, like the created dataset, axes information, legend and the “scatterOnClick” function. This function loads when clicking on a point in the chart the data connected to this point as first rows of the data table.

If the map or the chart data is returned before the table data, the respective part will be created with the creation of the view. Otherwise, a loading screen is shown in the space, where the component is placed until the data is received. When all these components are completely loaded, the page is presented as shown in Figure 4.6.

4.2 New functions and improvement of old functions

After migrating the homepage from AngularJS to Angular a big part of my work was the improvement of old functions and the implementing of new ones. After that transformation, the new version is a lot more stable.

4.2.1 Map related functions

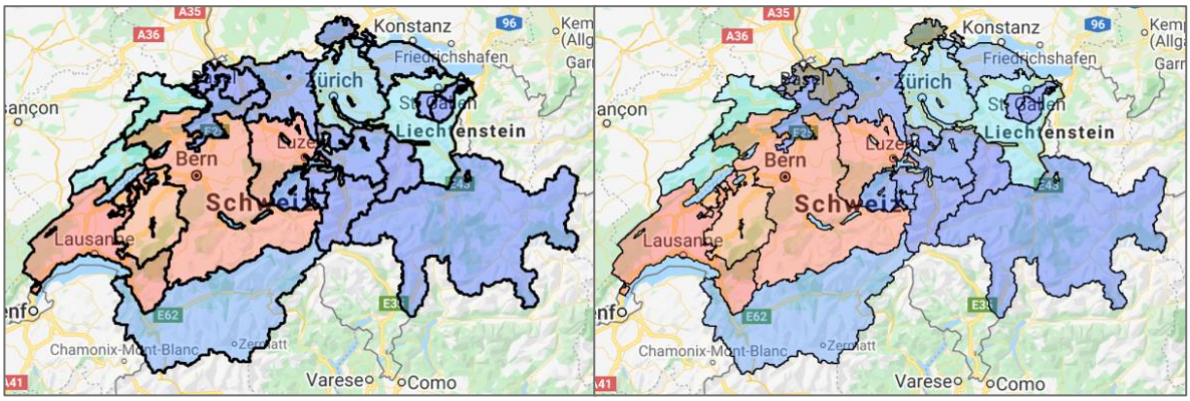


Figure 4.7- Comparison of canton overlay
(old: left, new: right)

The first improvement can easily be seen when loading the canton view of any query, like shown in Figure 4.7. In the old version, data were not processed correctly and so the overlay was computed twice, and both were put on the map. In the case of the canton view this was only a problem of the appearance, but in other cases, especially when changing fast many aspects of the map, this could lead to wrong overlays. Two examples of these problems can be seen in Figure 4.8. In the left picture, due to fast zooming one of the canton overlays is not cleared, while the other one is changed to the location data. In the right picture fast changes of density and regression option was made, which led to this erroneous picture.

In the revised version such pictures are no longer possible. The code part which led to multiple loading was corrected. A new overlay is no longer computed without deleting

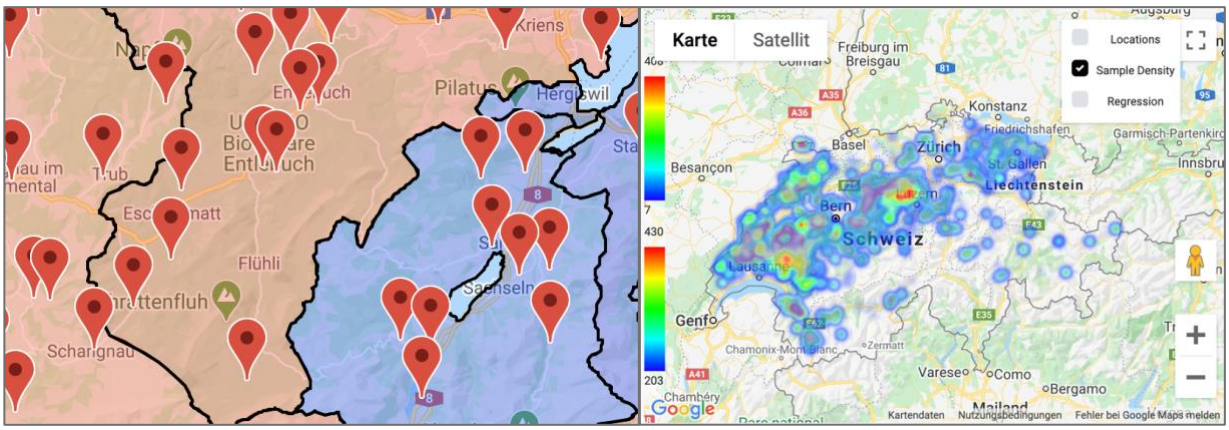


Figure 4.8 - Erroneous overlays
Left: location data and canton data shown together, but intension is to have one of them
Right: Regression data and density data in one map

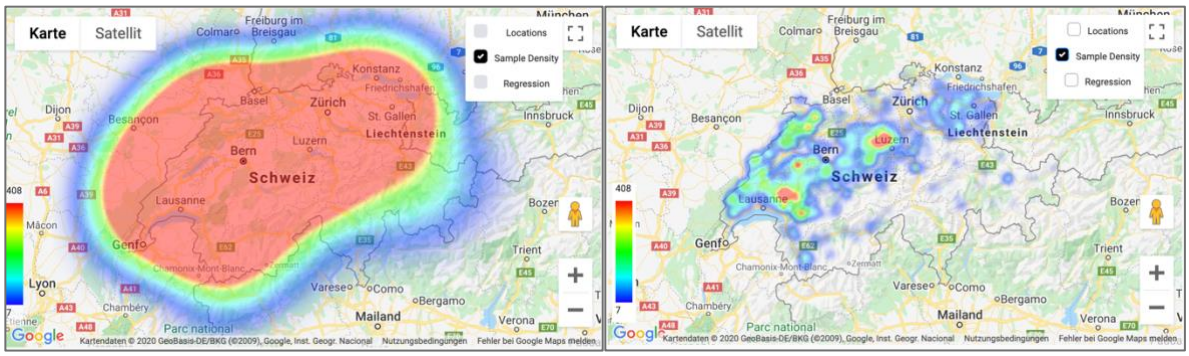


Figure 4.9 – Comparison of density overlays after zooming out (old: left, new: right)

existing ones. On the other hand, the zoom level as well as the selected overlay type are controlled just before creating the new overlay. If the overlay type of the request parameters doesn't correspond with the current overlay type, or the zoom level of the request parameters is not the zoom level of the map in case of location data, the overlay data is saved, but not loaded as overlay on the map. This way, it is no longer possible to have multiple overlays to be loaded at the same time.

One of the main problems, which resulted in creating this bachelor thesis, was the fact that the heat maps not always displayed correctly the sample and nutrient density at all zoom levels (Figure 4.9). The problem here was the incorrect implementation of the event listener. After one change of the zoom level, the event listener wasn't added anymore. That's why the overlay wasn't correctly computed anymore with the next zoom changes. The reason for this was, that the density, which was at the beginning computed correctly for the geographical distance, was computed in changing zoom levels according to the displayed distance. This resulted in a much higher density when zooming out and a lower density when zooming in.

Since data often are not linearly distributed, I introduced a new function for changing the colour gradient. By clicking on the colour legend in the bottom left edge of the map, the user opens the associated modal window. There, he can change the gradient with a slider. The value of the slider gives the exponential of the function, in which the colour is distributed. In the middle, the distribution is with exponent 1. The values upwards are 2 to 10, with step size 1, while the values downwards are $\frac{1}{2}, \frac{1}{3}$ and so on until $\frac{1}{10}$. With this change, the user can emphasize changes in high or low value region, while the change in the other value region is neglected.

This new function covered some problems with the visualization, but it remains a problem with the visualization of the data in high density locations. As the heatmap takes the highest value in a specific radius, lower values were covered by these high values and therefore, the heatmap was not as informative as it could be. For solving this problem, I introduced the possibility to change manually the radius in the map. This function is also placed in the gradient modal window. With this, the user can decide to which accuracy he

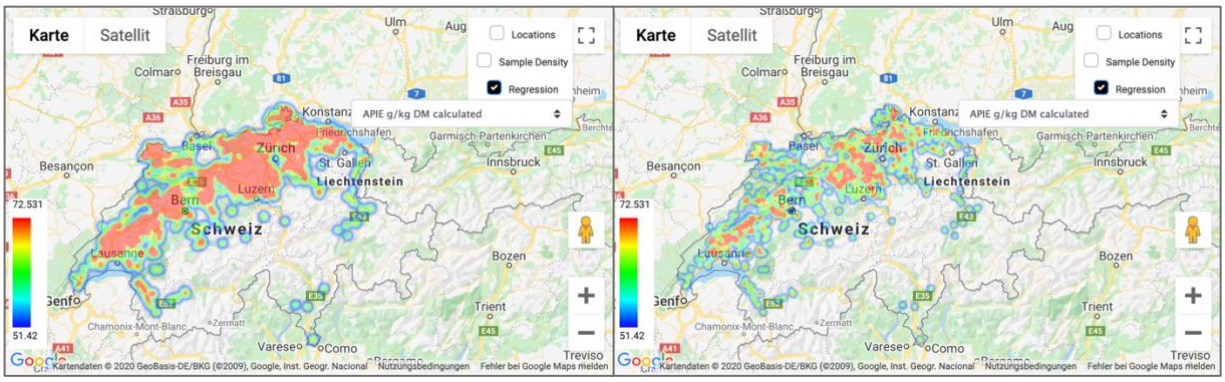


Figure 4.10 - Regression data of the "Corn silage survey" with different heatmap radius
Left: 7.5 km, Right: 5km

wants the overlay. An example of two different heatmap radius is presented in Figure 4.10.

Another new function is the possibility to have the map without an overlay. By selecting none of the overlay options, no overlay is loaded, while the existing one is cleared. This can help when someone wants to see the information of a standard map around some points, before he changes again to one of the overlays.

Figure 4.11 - Marker information

A new function, which was previously intended, but not yet implemented, is location information. By clicking on “Detail” in the info menu of a marker (Figure 4.11) the user can see on top of the table highlighted all information belonging to this point. The clicking starts a server request for table data with the selected location coordinates. The server will search the specified data and puts it on the top of the existing table data and sends it back to the browser. There, the table data will be updated and consecutively, the new table with the first rows containing the location’s data is loaded.

Another change is the speed of loading. Although the first loading of data takes still the same amount of time, every change back is now a lot faster. A request from the server is only done, when new data should be return. Before, aside from the canton data, every zoom change resulted in a new server request, although the returned data was the same. The other reason for the speed improvement is that the loaded data is saved in the cache. The drawback of this second reason is that there is more data in the cache, but the space occupation is not that much, that it becomes a problem for any computer. This will be further discussed in section 5.1.

4.2.2 Chart related functions

The first change was the updated colour picker (Figure 4.12). In the old version the possibilities for adjusting the colour were more limited than now. The user can now not only select it in the colour field, but also by giving the colour code. For this, he can choose between Hex, RGBA and HSLA.

Figure 4.12 - Colour Picker (left old, right: new)

An improvement was realized by the point selection in the scatter chart. While the detailed data was already put in the first row of the table for a single record, multiple records for one point were not shown due to an error in the SQL query. This is now working for any number of data at the same point.

Another improvement of the point selection in the scatter chart is the zooming in the table to the respective point, if only a single location refers. Previously, the zoom level was the same and only in the location view the map was moved to the point. But now the map zooms always to the point.

A general problem with the map occurred when all selected data were from the same date. The chart couldn't display this data, due to a division by zero. When calculating the position of the data point in the chart, the position was computed in relation to the minimal and the maximal value. But in case of only one value, the minimum and the maximum are identical and the difference therefore equal to zero. Dividing a value through this difference resulted in the above mention division by zero error. In the new version this problem is solved by dividing through one if only one value exists, since the position of the data point in this axis has no relevant information.

4.2.3 Table related functions

A small, but useful change are tooltips for the buttons and the filtering text boxes. It was not clear before how the filter should look like.

The filtering in general was also revised. Previously, it only filtered the data in the table. Now it also filters the data in the map and the chart. This can give a lot more insights from the data. The map can show how the filtered data points are distributed geographically and the chart reflects how in respect to other nutrients for example the data changes. These insights are not easily found only using a list of data. When a filter request is typed, the input is controlled by the frontend in difference to the previous version, where it was

```
options.tableRangeSearch.forEach((element: any, index: number) => {
 filter = filter + `,filteredByNutrient${index} AS (
  SELECT d_feed.feed_key AS feedkey,
  CASE (${element.min ? `MAX(reference_data.raw_value) >= ${element.min}` : ''}
          ${element.min && element.max ? ` AND`: ''}
          ${element.max ? ` MAX(reference_data.raw_value) <= ${element.max}` : ''})
          WHEN true then 1 ELSE 0 END AS is_element
  FROM reference_data, d_nutrient, d_feed
  WHERE
  d_nutrient.nutrient_key = reference_data.nutrient_fkey
  AND d_feed.feed_key = reference_data.feed_fkey
  AND d_nutrient.specie_id IS NOT NULL
```

Figure 4.13 - Code fragment belonging to the filtering of the reference data

controlled server-side. If the search has the correct format, the data requests of the three visualizations are started. The result of each request is shown as soon as it is received by the frontend.

Another part of the filtering, that didn't work with returning the complete information, was the information of erroneous filter instructions. Although the first erroneous input got a red border, followed by the fact that incorrect inputs were ignored. By checking all fields before sending the requests to the server, we can show all erroneous fields. This is in difference to the previous version, where the first incorrect field created an error on server side and this field became red bordered. Another benefit of the frontend control is the minimization of unnecessary server requests, especially in the new version with multiple requests for the different visualizations.

Last change related to filter is the implementation of the filter also to reference data. This functionality was not yet implemented, because filtering is more difficult than for the detailed data. The nutrients data in the last section of the created query, where the filtering is added in the detailed data, is in multidimensional PostgreSQL arrays converted. This leads to a more difficult filter query. The solution found is geared to the approach used for sorting. The code fragment belonging to this new part can be seen in Figure 4.13. The function, to which this fragment belongs, adds this part to the complete request with the other filter conditions. Each filtered column has an own view, distinguished by the index number. If the nutrient, which was filtered within the WHERE clause part, has its value in the selected range of the value filtering, the field with the name "is_element" is 1, otherwise 0. At the end of the query, the feed keys will be filtered by only selecting the keys with "is_element=1" in all "filteredByNutrient" views.

4.2.4 Upload improvements

When uploading new data from an excel file in the database, a frequent problem was the inability to upload data with apostrophes in it (e.g. in “Ensilage d'herbes”). This problem is solved by adding a second apostrophe just after the original one. This way, the query recognizes that this is a string apostrophe and not an end of the string, which is also marked by an apostrophe.

In the previous version it was not easy to have an overview by the error report. Too much irrelevant information was shown, and the relevant data had to be found by extended effort. This was minimized in the way that if an error occurred, only the information of the error rows is shown, and if everything worked well, the only information is that the information is uploaded successfully. Additionally, the error representation was standardized, starting with “ERROR on line xxx in the excel file:”, followed by the more detailed error explanation. In the earlier version, every correct inserted row gave the information that it was successfully inserted, which could lead to a very long error report.

4.2.5 General improvements

Not a function itself, but useful for the user is the addition of loading spinner over the parts that are loaded. Along with that, the user can see that data is about to change and where. Previously it was often not clear if the functions were started or not. This is given for each of the visualization parts as well as the start page and the upload. As soon as new data is loaded, a signal is sent to the respective visualization part, which starts the spinner. After loading the new data, a new signal is sent for informing that the data is now loaded. Since the computation of the visualization is fast, the removing of the spinner and the display of the new data is at the same time for the user.

Another improvement is the loading speed of the complete page. Previously, the table was loaded first and then the chart and the map. By making the three parts independent, it is now possible to load all three parts parallel. This way the maximum loading time of the whole page is the maximum time of one of the parts, not as before the table time and the longer one of the other two parts. For this, a first, fast request of the column data is made. This request is included in the query search for top queries. As soon as this information is loaded, the visualization parts load their respective data. The column data of the table is also relevant in the map (nutrient options for regression) and the chart (nutrients to be shown).

5 MEASUREMENTS

What is the impact of the coding changes regarding speed and size? How are the size of the requested data and the speed related? How much time need the subparts of a visualization process? The goal of this chapter is to find answers to these questions.

The sample data is taken on a MacBook Pro 2018 in Chrome, while the homepage was running on localhost. The database is hosted on feedbase.uzh.ifi.ch.

In general, the taken queries are the saved top queries. On one hand, as these queries were saved by admin users from Agroscope, we can assume that they have a relative high importance. On the other hand, we can assume that they are random enough to cover different part of the database and with that, the queries differ. Lastly, the reproduction of the results is more easily possible.

In this part only the summarized measurement results will be shown. For more detailed measurement, see Chapter A.2.

5.1 Loading speed of first visualization

As mentioned in section 4.2.5, the loading speed of the first visualization side has improved in comparison to the original AngularJS version. How much of this acceleration is due to the change of AngularJS to Angular? This information is interesting, since one of the reason for the change from AngularJS to Angular was the higher performance expected. How much of the acceleration is due to the change from mostly serial loading to the parallel loading?

Another aspect of the loading of the first visualization, beside the complete loading time for all visualization parts, is, how long it takes for the first view to be loaded. The meaning of this is, how long does it take until the user sees at least one part of the visualization. As the visualization is loaded as soon as the table data is returned from the server. The time for loading the first view is equal to the time until the table data is loaded. By measuring this first view loading time, we pinpoint the influence of the changed loading plan.

The measured data in this section are evaluated in 3 different environments.

1. The original homepage: The original homepage is the homepage as it was when the bachelor thesis started. It is written in AngularJS.
2. The original Angular homepage, called Angular in the graphs: This homepage is more or less the same as the original version, but now migrated from AngularJS to Angular.
3. The final homepage called New Version in the graphs: This is the version with the speed improvements due to different coding of several parts.

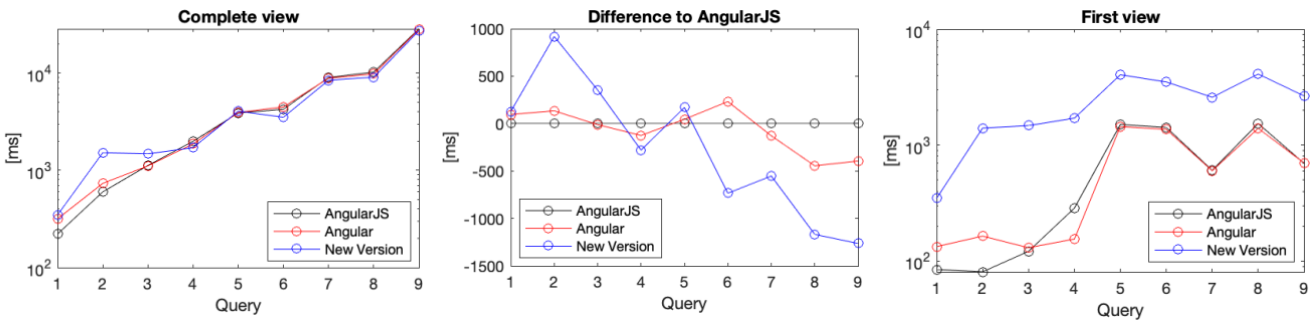


Figure 5.1 - Speed of the first visualization
The queries are ordered by the total loading time of the AngularJS version
Left: The time needed for loading all parts of the visualization (logarithmic y axis)
Middle: Difference of the total loading time of AngularJS to the following implementation steps
Right: Loading time of the first view, meaning the change to the general visualization view

In the original homepages (1 & 2) the parameters were loaded first, then the table data (with paginate) and at last the map and chart data. In the new version, we first load the params and the column data and then in a second step the data of the three visualizations. The column data, which is loaded now with the params, is used in all visualizations. Since this is more data, the request takes longer, but the same data isn't loaded anymore after in the table data request, which improves the speed of this request later on. Although the first request takes longer, the whole data loading process takes less time, since the three other, usually longer requests are now parallel.

When we focus on the change of AngularJS to Angular, we can see in Figure 5.1 (detailed data in Table A.1 in the appendix), that the speed of Angular is still very similar to the speed of AngularJS. The trend is, that it goes minimal faster for longer requests, but the difference is not big. The change resulting from the parallelization is bigger. The time has risen for fast requests, but for longer requests, the whole time takes faster. For example, the longest request, which took in AngularJS 28.4 seconds and 28 seconds, takes now with the parallelization 27.1 seconds.

A drawback of the parallelization is the generally risen time for loading the first page. One reason for this is, that the column information is loaded with the params and not parallel with the row data. This can increase the needed time. Another possible reason is, that the parallel requests slow the table request down.

The reason, that the column information is loaded with the params and not parallel with the row data, is probably also responsible for the longer total loading time of small requests.

5.2 Cache size for map data

In the old version, only the visible data is loaded and saved in the cache. For faster visualization of the data, the new version keeps data once loaded in the cache, as long as no query or radius change is asked. This way, comparison of different overlays is faster and with that user friendlier. But how much data have to be cached? This question will now be discussed.

We will use two different sizes in this section, the transferred size and the resource size. The transferred size is the compressed size of the data, which the user loads, while the resource size is the uncompressed size.

If every map overlay is loaded, there are 4 different data sets, the location data (markers), the canton data, the density map and a regression map.

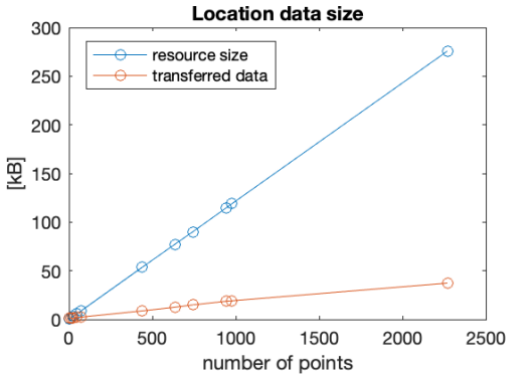


Figure 5.2 - Data size of location data

First, we examine the size of the location data. The size grows linearly, as one can easily see in Figure 5.2 (tabled data in detail in Table A.2 in the appendix), but the space of the data is relatively small. Even for 2271 data points, the amount of data points of the hay survey, the resource size is still only 276 kB, transferred in 37.3 kB.

The canton data size depends on the number of polygons. Since the maximal number of these polygons is the number of cantons, the number has a fixed upper limit. This size limit is 2.5 MB for the transferred data and 6.3 MB for the total resource size. The size of the polygons depends on the shape of the individual cantons. The largest canton in this respect is Bern, which has a transferred size of 261 kB and an actual resource size of 651 kB.

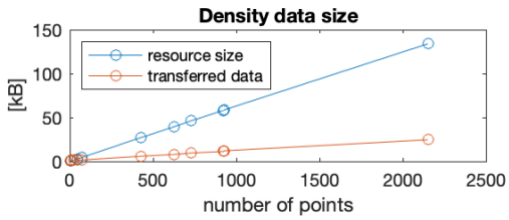


Figure 5.3 - Data size of density data

The density data size grows linearly, like the location data, but with a smaller factor (Figure 5.3, tabled data in detail in Table A.3 in the appendix). For 2153 density data points, the resource size is 134 kB, transferred in a compressed size of 24.6 kB.

Also growing linearly, but with the highest factor is the data size of the regression data (Figure 5.4, tabled data in detail in Table A.4 in the appendix). 2153 data points need 333 kB space, transferred in 63.2 kB. This data has always a constant data included, namely the statistics of the regression.

By summarizing this data sizes, we can see that the data size is not that relevant for time spent. For the top query with the highest amount of data, the "Hay survey regional NEL content 2005-2017", we have 6577 data points and all canton polygons, which has a size of 743 kB for the data points and 6.3 MB for the canton data, which results in a total data size of 7.043 MB.

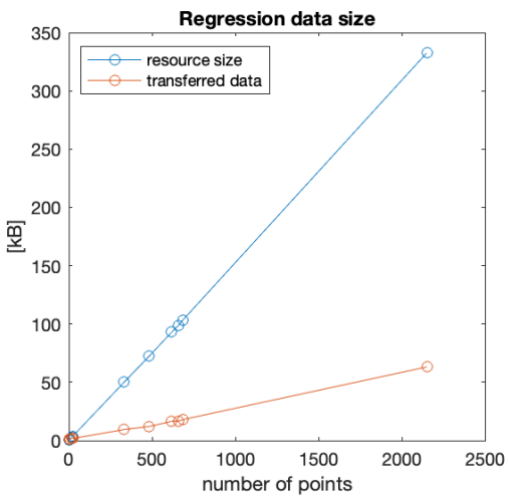


Figure 5.4 - Data size of regression data

5.3 Connection of various loading times

Where comes the different duration of the requests from? Is it the number of entries in the database connected to the request? Is it related to the number of returned points and/or the size of the data? To answer these questions, I measured all main request duration.

First, the parameter request is examined. Here, we can check the connection of the size to the number of rows to be returned, as we haven't done this yet in the last section. Following, we can see, that the resource size rises with growing number of points, the transferred size stays stable at 614 B.

When examining the duration of the parameters, we see, that the download time is around 0.5 ms. This is probably due to the not changing transferring size. The delay and accordingly the connected total download time is mostly arbitrary, as no obvious connection to another measurement could be found. This small time has no relevance in comparison to other data request that follows.

Examination of the chart data reveals, that the waiting and thereby total loading time grows with the number of entries in the database (Figure 5.5). Although the download time grows loosely with the number of points, it is mostly irrelevant in comparison to the waiting time for the server. For example, for 1060 points in the scatter chart we waited in average 8500 ms, while the download took only 0.724 ms in average. That the size of the data grows linearly with the number of points is similar as the map data size, which was shown in the last section.

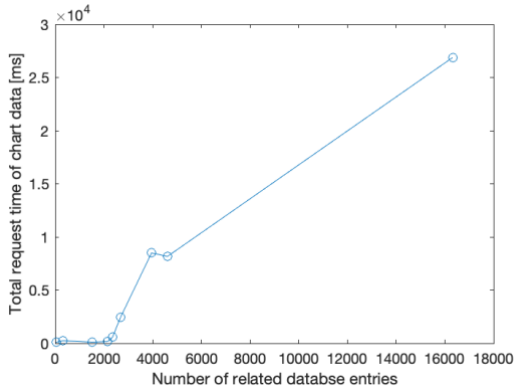


Figure 5.5 - Request time of chart data vs number of related entries in database

The size of the table data changes with the size of the column data. Here, the total number of to the query related entries is not relevant for the size, since only 50 entries are returned at a time. Although the size of the table data varies, the download time cannot be associated to the data size as it stays around 0.5 ms. For a very small number of related table entries, the request time is small, but as soon as the number of entries is over 2000, the total request time for table data stays around 3 seconds. Even for 16326 related entries of the "Hay survey regional NEL content 2005-2017" query, the total time is 2.47 seconds. But this isn't the highest needed time. For example, needs the request "Corn silage survey", which has 3956 entries, clearly longer with 3.8 seconds.

The size of the canton data is generally related to the number of cantons to show. The existing differences are due to different sizes of the polygons of the cantons. Like previously explained, we have large cantons like Bern with 651 kB and tiny cantons like Appenzell Innerrhoden with 56.9 kB. As only the summarized entry data is shown in the

canton view, the loading time doesn't change much with growing number of entries in the database. Requests, which return most of the cantons, need generally around 2 seconds, independent of the number of entries.

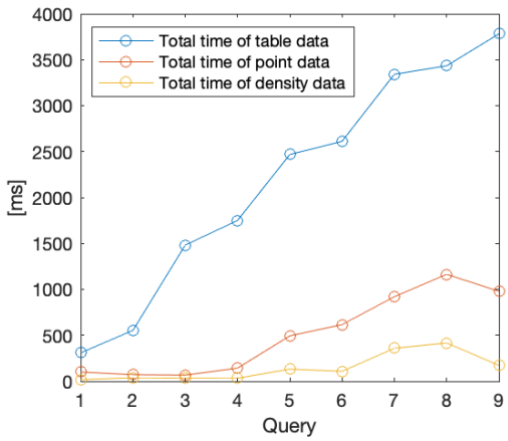


Figure 5.6 - Comparison of the, probably with filtering connected, total loading times Ordered by rising total time of table data

The download time of point data is again rising with the growing number of points, but on a small scale. Even for 2271 points, the download time in this measurement was in average 2 ms. The waiting and total request time grows similarly to the table data. This leads to the assumption, that the time of both could be related to the time needed in the server for filtering the data. The higher time demand of table data in comparison to point data is due to more related information that has to be loaded with each data point.

For the density data, the number of points is not relevant for the request duration. But it has a lot similarity with the request time of the point and table data. This leads to the assumption, that it is as well related to time needed for the filtering. That these three parameters have similar growing behaviour can be seen in Figure 5.6. The download time is again, in small scale, dependent from the number of points. But even the largest tested query ("Hay survey regional NEL content 2005-2017"), which had 2153 data points, only needed 0.84 ms for the download.

The request time of the regression data is dependent on the number of points. Interestingly, but without an explanation, is the download time for a big number of data points. Although the other download times were similar to all other download times, meaning there where under 1 ms, the biggest returned request, which is the query "Hay survey regional NEL content 2005-2017" with 2153 returned data points, takes clearly longer with 5.8 ms. Despite this, the download time is still a small part of the total time, which is in this case over 14 seconds.

Although the number of points in a scatterplot and the number of data points in a map (location density or regression data) grow in general with the number of entries in the database, the grow of elapsed time slows down with growing number of entries. The reason for this finding is related to more entries in the database, the more likely it is, that multiple entries have the same point in the chart or map.

In summary, the following can be observed:

- The download time is a very small part of the request time
- The waiting time and total request time aren't related to the size of the return data in general.

5.4 Initialization of visualization parts

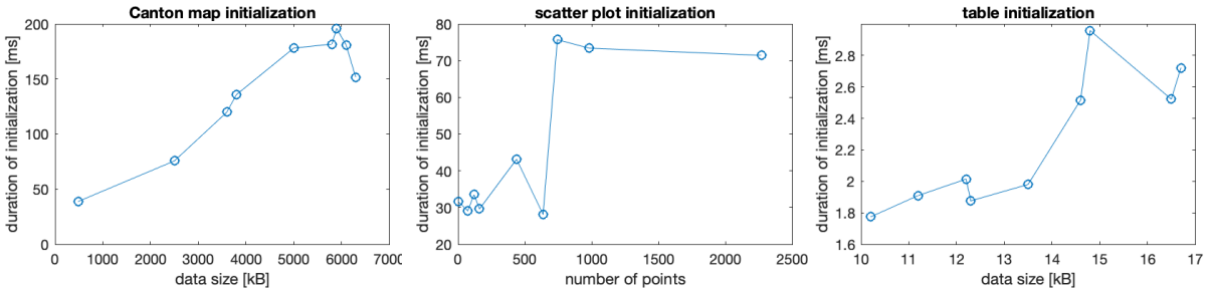


Figure 5.7 - Duration of initialization
(left: Canton map, middle: Scatter plot, right: Table)

Although the initialization of the different visualization parts is not a major part of the whole visualization process, it can be interesting to analyse them more in detail. By examining the total duration of the initialization, we can see, that the time is related to the size of the data for the canton and the table initialization. The scatter plot initialization has not really a connection between these two sizes.

Another observation is the fact that the scatterplot and the table need very little time for the whole initialization. For the table, initializing the whole table needs less than 3 ms. The scatter plot was in all measurements created in less than 80 ms, even for as many points as 2271. The initialization of the canton map took longer than the previous two, but with 207 ms the longest measured duration when almost all cantons have to be loaded, the duration is still not long, especially if we compare it with the request loading time in this case of around 3 seconds.

But which part of the initialization takes which amount of time? This is the underlying question for the next subsection. As the initialization of the table is very fast, this visualization part is not further examined.

5.4.1 Initialization of the map

For analysing the initialization time of the map, I divided the map initialization process in the following parts:

- The initialization of the map
    - This covers the creation of the google maps map with Switzerland in its centre as well as the initialization of the overlay map without any data yet
- The adding of the zooming listener
    - This listener checks, if the zoom level, which is at first 7, may get over 9. If this happens, the map should load the location data
- The adding of the overlay
    - This is the process of adding the canton data to the overlay map

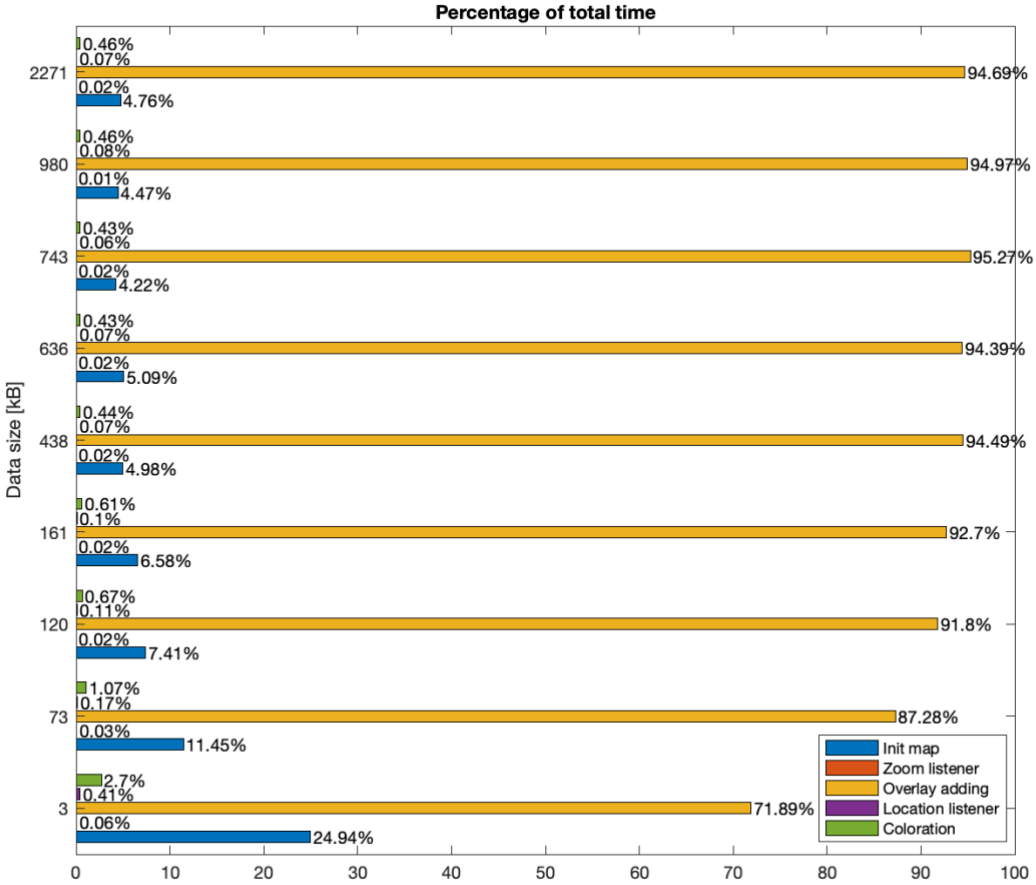


Figure 5.8 - Percentage of the different parts of the map initialization

- The adding of the location listener
  - This listener is responsible for displaying the number of data points in a specific canton, when the user clicks on it
- The coloration
  - This means the process of dyeing the canton polygons with the colour responding to the amount of data points in it.

The analysis shows, most of the needed time is consumed for adding the overlay. The time needed for it grows with the data size. Since the other tasks are more or less constant for different data sizes, the needed time in relation to the others expands. But even for the smallest measurement the timely task of this part needs 72% of the whole time. This grow participates in 95% of the time, as we can see in Figure 5.8.

The rest of the time is dominated by the initialization of the map, which takes for any data size around 9 ms, which goes from 25% from the smallest measure to around 5% of the time for extended measurements.

The coloration takes around 0.83 ms and is not related to the size of the data. The needed time is for the smallest measure 3% of the whole time. For higher measurements, the needed time is under 0.5%.

Table 5.1 - Average time for the different map initialization parts

size [kB]	Init map [ms]	Zoom listener [ms]	Overlay adding [ms]	Location listener [ms]	coloration [ms]	Total (computed) [ms]
483	**9.19**	**0.022**	**26.492**	**0.15**	**0.996**	36.85
2500	**9.76**	**0.022**	**74.37**	**0.146**	**0.912**	85.21
3600	**8.99**	**0.022**	**111.44**	**0.128**	**0.814**	121.394
3800	**8.888**	**0.026**	**125.276**	**0.138**	**0.818**	135.146
5000	**8.888**	**0.03**	**168.538**	**0.128**	**0.786**	178.37
5800	**9.118**	**0.042**	**169.222**	**0.1302**	**0.772**	179.2842
5900	**7.942**	**0.03**	**179.308**	**0.12**	**0.814**	188.214
6100	**7.858**	**0.024**	**166.912**	**0.142**	**0.816**	175.752
6300	**8.108**	**0.03**	**161.434**	**0.124**	**0.786**	170.482

Almost no time is needed for adding the zoom and location listener, which both use in general less than 0.1% of the whole time needed. The zoom listener needs in average 0.03 ms and the location listener 0.13 ms.

5.4.2 Initialization of the chart

For analysing the initialization time of the chart, I divided the scatter chart initialization process in the following parts:

- The adding of the tags
    - This saves the nutrients, for which the scatter chart should be created, and adds a random colour to it
- The processing of the json file
    - This means adding the data points to the chart
- The actual initialization of the chart with all functionalities
    - This means the creation of the visible chart with all its functionalities
- The actual initialization of the chart without most of the features
    - The removed features are the opening of the colour picker when clicking on the colour box and the loading of the data to the point in the table. That the information to each point is shown when hovering over it, couldn't be removed without additional code, because this is a standard function of these charts
- The features
    - The features, which were previously mentioned.

The initialization of the chart without the events was not measured in the same turn with the other measurements, because this part is created in one predefined function of the chart library and can therefore not be separated in the same turn. Additionally, the legend of the points when hovering over the points was not removed, because this is an integrated functionality of the chart.js library, which I would have to remove in an additional step, which asked for more time, therefore it was omitted.

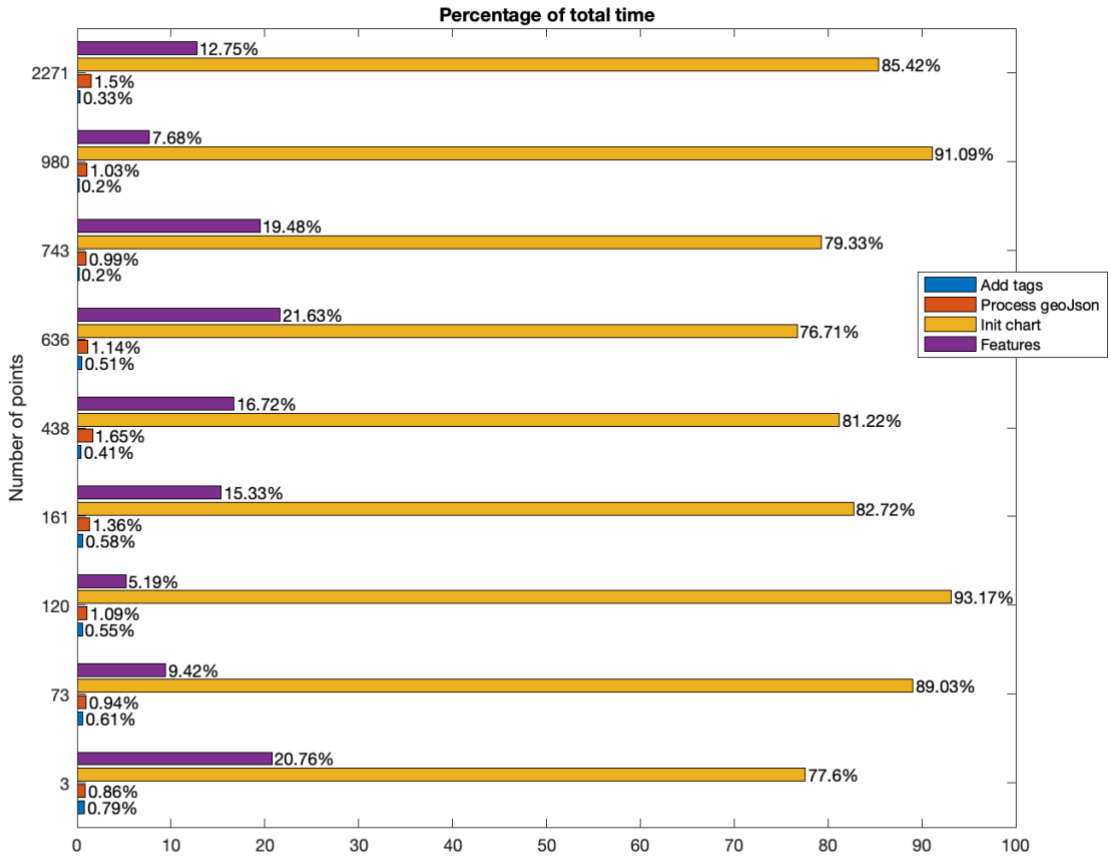


Figure 5.9 - Percentage of the different parts of the chart initialization

Most of the whole initialization time was, in difference to the map initialization process, used for the initialization of the chart. The needed time for this part is around 84% of the complete time, as one can see in Figure 5.9. The processing of the data from the json file takes only around 1% of the time. What the initialization of the chart and the processing of the data have in common is, that for both of their duration grows with the number of data points. The duration of the map initialization grows from 22.3 ms to 69.8 ms, while the processing of the geoJson only needs between 0.25 ms to 0.79 ms. The average times of the measurements can be seen in Table 5.2.

Table 5.2 - Average time for the different chart initialization parts

points	**Add tag [ms]**	**Process geoJson [ms]**	**Init chart [ms]**	**Features [ms]**	**Total (calculated) [ms]**
3	0.226	0.246	22.308	5.97	28.748
73	0.164	0.252	23.948	2.53	26.898
120	0.172	0.338	28.986	1.61	31.11
161	0.164	0.382	23.252	4.31	28.108
438	0.154	0.618	30.444	6.27	37.482
636	0.156	0.348	23.32	6.58	30.4
743	0.168	0.832	66.67	16.37	84.04
980	0.156	0.79	69.844	5.89	76.68
2271	0.166	0.754	42.824	6.39	50.134

Beside the initialization of the chart itself, the features on the map need also a good amount of the total initialization time. The amount varies from query to query in a not explainable behaviour from 5.2% to 21.6%. It is not explainable, as it has no connection to the number of points or any other item.

The adding of the tags also need always less than 1% of the time. The needed time for this is more or less constant of around 0.17 ms.

6 DISCUSSION

6.1 Colour gradient in map

The new possibility of changing the colour gradient helps with the visualization of not linear distributed values. The user can put the focus on the data value parts, where the most interesting changes happen. The other possible change of the heatmap radius solves the problem with regions, where the density is high. With these options, the usability of the map was improved.

What was the reason to not doing the radius selection it adaptive by itself? If the radius changes from visualization to visualization, it complicates the comparison of different maps. By giving the user the possibility to change it himself, he has the knowledge of the used radius.

6.2 Loading speed of the first page

We have seen, that the loading of the whole page, meaning all visualization parts, became faster for longer requests with the parallelization. The drawback is a generally longer loading time of the first view, meaning the time, until the table data is loaded. As mentioned in 5.1, reasons for this are on one hand, that the column information is loaded with the params and not parallel with the row data, and on the other hand, that the parallel requests slow the table request down.

The minimal effect of the change from AngularJS to Angular with same coded order is probably because most of the needed time is the SQL request in the PostgreSQL database, on which the migration has no effect.

What can be possible done to improve this? An intention can be to decouple the column data request and the parameter request. This way, the table data could be loaded parallel to the column data as soon as the parameter data is returned. The map and chart data would wait for the column data as well until they start. A drawback of this solution is, that the server requests in short time rises further. Since the parallel loading can already have an effect on the loading time for a smaller number of requests as just mentioned, this would lead to even longer waiting time, especially if not only one user is on the page at the same time. Since the measuring of the duration with many requests at the same time wasn't possible for me, I haven't followed this question any further.

6.3 Data size

In Chapter 5.2, we measured the size of the map data. We saw, that the data, except for the canton data, grows linear. To what extent can this size rise? For answering this, we have to find a realistic upper bound for the number of data points. As the database consists of data from agriculture, a possibly good upper bound is the number of farms in the country. In the year 2019, the number of them was 50'038 (Bundesamt für Statistik,

2020). If we calculated for this number the expected size, we get 16'958.3 kB for the linear growing data. This results in a size of approximately 22.3 MB, when adding the canton data. In this calculation, it is assumed that every location point creates three different data points, one for each location data, density data and regression data. As we can see, the data size is still very small under this condition.

Why can we assume that there won't be many more data points in the map under the assumption of the number of farms? Data from the same location are summarized in one data point in the map and the size change of a point can be neglected. That's why the number of points can't grow unlimited.

6.4 Connection of various loading times

As the examination of the loading speed has shown, the time for filtering is still a major part of the request time. Therefore, the improvement of this filtering is still subject for improvement. The time for downloading is generally small in comparison with the amount of visible data. The meaning of this is, that although we have many points visible for example in a chart, the data size and the download time is not high.

Problematic for scatter charts are requests, where many entries from the database have to be processed to returned data for the chart. This can lead to very long request times.

6.5 Initialization of the visualization parts

We have seen in the measurements, that although the length of the initialization is dependent from the data size, the speed of the initialization is in general very fast. The library is already that far optimized, that the time of the initialization is even for much data very small. Adding of additional features is therefore also not a problem regarding the speed. The gain of these features will be by far greater than the loss of time.

6.6 Future work

As we have seen, the waiting time of requests for new data can take a very long time. This diminish the user experience. That's why this is still a very important topic to improve.

This thesis has put its focus on the improvement and measuring of the speed in the frontend. But most of the needed time is due to waiting for the request. There, most of the time is lost to the time needed for the SQL query request in the PostgreSQL database. Probably, these requests can be accelerated in the future.

As mentioned in 6.1, although the user has now the possibility to change colour gradient of the heat map, there is still sometimes the problem, that data is not shown differentially enough in regions with high data density. This can still be improved for better usability of the map.

BIBLIOGRAPHY

Agroscope. (2020). *Agroscope*. (Agroscope) Retrieved May 31, 2020, from https://www.agroscope.admin.ch/agroscope/de/home/ueber-uns/agroscope.html

Anderson, M. (16, July 27). *digitalocean.com*. Retrieved April 9, 2020, from https://www.digitalocean.com/community/tutorials/how-to-move-an-apache-web-root-to-a-new-location-on-ubuntu-16-04

Angular. (2010, October 21). *Releases*. (Google) Retrieved June 1, 2020, from https://github.com/angular/angular.js/releases?after=v0.9.4

Bracher, A., & Boltshauser, M. (2013, August). *Gebrauchsanleitung Schweizerische Futtermitteldatenbank.* Retrieved May 31, 2020, from https://www.feedbase.ch/layout_resources/lng/help_de.pdf

Bundesamt für Statistik. (2020, May 11). *Landwirtschaftsbetriebe, Beschäftigte, Nutzfläche nach Kanton.* Retrieved June 4, 2020, from https://www.bfs.admin.ch/bfs/de/home/statistiken/land-forstwirtschaft/landwirtschaft/strukturen.assetdetail.12687403.html

Fenton, S. (2014). *Pro Typescript.* Apress.

Google. (2020). *Angular Material*. (Google) Retrieved June 9, 2020, from https://material.angular.io/components/categories

Google. (2020). *Introduction to Angular concepts*. (Google) Retrieved 6 1, 2020, from https://angular.io/guide/architecture

Google. (2020). *Modules*. (AngularJS) Retrieved June 1, 2020, from https://docs.angularjs.org/guide/module

Google. (n.d.). *Tour of Heroes app and tutorial*. (Google) Retrieved June 9, 2020, from https://angular.io/tutorial

JetBrains. (2020). *WebStorm*. (JetBrains) Retrieved June 4, 2020, from https://www.jetbrains.com/webstorm/

Manjunath, M. (2018, April 6). *AngularJS and Angular 2+: a Detailed Comparison*. Retrieved June 1, 2020, from https://www.sitepoint.com/angularjs-vs-angular/

A APPENDIX

A.1 Server Setup

For testing my website on an external server, I got a new ubuntu server from the university with previously installed apache web server, which showed the initial apache page. My project was loaded to the ubuntu server with git in the home directory (as directory called feedbase4).

From here, I had to set up my website. For completeness, it will be explained at this place. A great help were the instructions from Melissa Anderson on digitaloceans.com. (Anderson, 16)

Since apache has no access to the feedbase4 directory, I had to choose a place accessible for apache, where I would clone the project as soon as it is built. Therefore, I created a directory feedbase in the directory with path /var/www/. In this directory, the default apache homepage is also saved. For simplicity from building the project, the path for the new index.html file became /var/www/feedbase/dist/browser.

Next, I had to change the document root to the new folder with the command:

```
sudo rsync -av /var/www/html /var/www/feedbase/dist/browser
```

Next I had to edit the 000-default.conf file in /etc/apache2/sites-enabled/ by adjusting "DocumentRoot" to the new root. The same had to be done in the file default-ssl.conf at path /etc/apache2/sites-available/.

After that, Apache had to be restarted by the command:

```
sudo systemctl reload apache2
```

A.2 Detailed Measurements

Following, the detailed measurements for confirmability are described.

A.2.1 Loading time of first visualization

In the new version, params also includes the column data. The time until first view is the needed time, until the table data is loaded, which starts the loading of the visualization view. The total time needed is the time, which the whole request took, i.e. what time was needed until the data of all elements of the visualization view were loaded.

Table A.1 - Loading time of first visualization (bold = average)

	AngularJS		**Angular**		**New Version**	
	First view	Total view	First view	Total view	First view	Total view
Corn silage survey	1507	10207	1571	9671	4195	9015
	1507	10077	1358	9798	4180	9130
	1547	10387	1388	9578	4143	9013
	1547	10207	1327	9597	4125	9105
	1577	10227	1337	10237	3903	9003

	1537	**10221**	**1396.2**	**9776.2**	**4109.2**	**9053.2**
Hay survey 2018	1366	4295	1367	4387	3491	3491
	1488	4468	1416	4646	3592	3592
	1407	4257	1337	4567	3449	3449
	1468	4238	1407	4367	3630	3630
	1387	3957	1317	4407	3383	3383
	1423.2	**4243**	**1368.8**	**4474.8**	**3509**	**3509**
Mountain hay > 1000 m, 2005-2011	1546	3936	1507	4057	4021	4021
	1469	3919	1478	3848	4118	4118
	1488	3808	1397	3867	3967	3967
	1457	3827	1377	3977	4095	4095
	1578	3918	1477	3887	4056	4056
	1507.6	**3881.6**	**1447.2**	**3927.2**	**4051.4**	**4051.4**
Hay 1st cut sugar regional pat	586	8916	618	8658	2570	8430
	606	8996	596	9056	2527	8357
	596	8976	600	8640	2639	8519
	589	9019	606	8926	2598	8408
	655	8985	589	8969	2605	8415
	606.4	**8978.4**	**601.8**	**8849.8**	**2587.8**	**8425.8**
Hay survey regional	687	28237	781	28661	3009	27619
	712	28442	663	28203	2509	26969
	689	28499	670	28270	2575	26845
	705	28405	663	28253	2615	27155
	685	28415	736	26636	2591	27091
	695.6	**28399.6**	**702.6**	**28004.6**	**2659.8**	**27135.8**
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	77	534	468	1079	1667	1667
	81	604	83	638	1714	1714
	89	650	85	640	1710	1710
	83	612	84	647	193	758
	74	594	101	661	1736	1736
	80.8	**598.8**	**164.2**	**733**	**1404**	**1517**
Sunflower seeds HO	78	181	261	498	369	369
	84	229	112	321	320	320
	90	250	94	278	418	418
	77	248	85	268	314	314
	94	198	115	220	318	318
	84.6	**221.2**	**133.4**	**317**	**347.8**	**347.8**
Mais kernels	819	2539	148	1928	1644	1644
	145	1815	157	1897	1760	1760
	167	1897	154	1844	1688	1688
	153	1893	156	1826	1821	1821
	148	1808	160	1830	1639	1639
	286.4	**1990.4**	**155**	**1865**	**1710.4**	**1710.4**
Barley	109	1061	112	1070	1481	1481
	123	1021	140	1108	1535	1535
	123	1088	146	1133	1388	1388
	127	1347	127	1177	1488	1488
	125	1086	127	1058	1480	1480
	121.4	**1120.6**	**130.4**	**1109.2**	**1474.4**	**1474.4**

A.2.2 Data sizes

Table A.2 - Data size of the location data in maps

Source	Number of entries	Resource size (kB)	Transferred data (kB)
Sunflower seeds HO	3	0.364	1.1
Fodder beets, fibre fractions	5	0.593	1.3
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	11	1.3	1.1
Mais kernels	26	3.1	1.4
Triticale, DE pig	48	5.8	1.9
Barley	73	8.8	2.3
Mountain hay >1000 m, 2005-2017	438	53.4	8.7
Hay survey 2018	636	77.3	12.5
Corn silage survey	743	90.1	14.9
Grass silage survey	946	115	18.7
Hay 1st cut: sugar regional pattern and correlation with ADF	978	119	19.1
Hay survey regional NEL content 2005-2017	2271	276	37.3

Table A.3 – Data size of the density data in maps

Source	Number of entries	Resource size (kB)	Transferred data (kB)
Sunflower seeds HO	2	0.129	0.823
Fodder beets, fibre fractions	4	0.256	0.951
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	10	0.627	1.3
Mais kernels	24	1.5	1.1
Triticale, DE pig	47	2.9	1.4
Barley	70	4.4	1.6
Mountain hay >1000 m, 2005-2017	425	26.9	5.8
Hay survey 2018	627	39.5	7.9
Corn silage survey	731	46.3	9.5
Hay 1st cut: sugar regional pattern and correlation with ADF	924	58.1	11.3
Grass silage survey	926	58.5	12
Hay survey regional NEL content 2005-2017	2153	134	24.6

Table A.4 - Data size of the regression data in maps

Source	Number of entries	Resource size (kB)	Transferred data (kB)
Sunflower seeds HO	2	0.45	1.1
Fodder beets, fibre fractions	3	0.575	1.3
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	9	1.5	1.2
Mais kernels	18	2.9	1.6
Triticale, DE pig	19	3	1.6
Barley	22	3.4	1.7
Mountain hay >1000 m, 2005-2017	332	50	9.4
Hay survey 2018	483	72.4	12.1
Corn silage survey	617	92.9	16.4
Grass silage survey	659	98.8	16.6
Hay 1st cut: sugar regional pattern and correlation with ADF	683	103	17.9
Hay survey regional NEL content 2005-2017	2153	333	63.2

Table A.5 - Summarized data sizes

Source	Number of data points	data size
Sunflower seeds HO	7	0.943
Fodder beets, fibre fractions	12	1.424
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	30	3.427
Mais kernels	68	7.5
Triticale, DE pig	114	11.7
Barley	165	16.6
Mountain hay >1000 m, 2005-2017	1195	130.3
Hay survey 2018	1746	189.2
Corn silage survey	2091	229.3
Grass silage survey	2531	272.3
Hay 1st cut: sugar regional pattern and correlation with ADF	2585	280.1
Hay survey regional NEL content 2005-2017	6577	743

A.2.3 Loading time

The total time is not the sum of the waiting time and the content download alone. Other tasks, which are not taken into the table are the time for queueing the resource, stalling of the connection start and the time of request to be sent. Additionally, it was counted for times, which are longer than a second, but only have an accuracy of 10 ms.

Table A.6 - Loading time part 1 - Parameters and Chart Data

Source		**Parameters**					**Chart**				
	Nr.	Size [kB]	Rows	Waiting [ms]	Download [ms]	Time total [ms]	Points	Size [kB]	Waiting [ms]	Download [ms]	Time total [ms]
Corn silage survey	1	1.9	15	597.43	0.48	600.71	1060	160	8490	0.68	8500
	2	1.9	15	532.58	0.57	534.63	1060	160	8440	0.68	8440
	3	1.9	15	510.26	0.59	512.25	1060	160	8510	0.79	8510
	4	1.9	15	533.27	0.47	535.91	1060	160	8480	0.74	8490
	5	1.9	15	523.65	0.46	526.19	1060	160	8550	0.73	8560
	Avg	**1.9**	**15**	**539.438**	**0.514**	**541.938**	**1060**	**160**	**8494**	**0.724**	**8500**
Hay survey 2018	1	2.4	20	4493.95	0.6	496.84	229	31.8	567.4	0.65	589.32
	2	2.4	20	501.93	0.56	504.43	229	31.8	551.31	0.5	555.83
	3	2.4	20	488.5	0.46	490.71	229	31.8	550.23	0.5	556.63
	4	2.4	20	581.52	0.39	584.28	229	31.8	573.45	0.49	577.8
	5	2.4	20	540.34	0.38	542.11	229	31.8	579.24	0.5	583.42
	Avg	**2.4**	**20**	**1321.25**	**0.478**	**523.674**	**229**	**31.8**	**564.33**	**0.528**	**572.6**
Mountain hay > 1000 m, 2005-2017	1	2.9	25	667.58	0.44	670.43	699	103	2390	0.77	2390
	2	2.9	25	660.75	0.39	663.39	699	103	2650	1.34	2650
	3	2.9	25	621.21	0.57	623.3	699	103	2370	0.58	2370
	4	2.9	25	646.22	0.67	648.5	699	103	2360	0.59	2370
	5	2.9	25	602.27	0.47	604.29	699	103	2340	0.58	2350
	Avg	**2.9**	**25**	**639.606**	**0.508**	**641.982**	**699**	**103**	**2422**	**0.772**	**2426**
Hay 1st cut: sugar regional pattern and correlation with ADF	1	0.838	4	282.45	0.58	285.01	980	154	8250	0.69	8260
	2	0.838	4	273.66	0.53	276.7	980	154	8140	0.97	8140
	3	0.838	4	240.31	0.52	243.13	980	154	8150	0.75	8150
	4	0.838	4	280.84	0.57	283.48	980	154	8080	0.72	8090
	5	0.838	4	242.48	0.6	245.28	980	154	8230	0.77	8230

	Avg	**0.838**	**4**	**263.948**	**0.56**	**266.72**	**980**	**154**	**8170**	**0.78**	**8174**
Hay survey regional NEL content 2005-2017	1	0.978	1	144.63	0.58	148.59	788	126	26910	0.62	26920
	2	0.978	1	170.01	0.67	184.72	788	126	26860	0.71	26870
	3	0.978	1	174.17	0.55	176.91	788	126	26830	0.63	26840
	4	0.978	1	172.87	0.27	175.08	788	126	26850	0.64	26850
	5	0.978	1	168.65	0.53	172.244	788	126	26990	0.74	26990
	Avg	**0.978**	**1**	**166.066**	**0.52**	**171.5088**	**788**	**126**	**26888**	**0.668**	**26894**
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	1	0.826	3	47.89	0.66	49.36	120	18.7	340.95	0.66	354.98
	2	0.826	3	63.22	0.57	64.63	120	18.7	195.44	0.54	209.42
	3	0.826	3	23.77	0.61	28.5	120	18.7	315.03	0.72	335.35
	4	0.826	3	51.6	0.54	56.1	120	18.7	175.76	1.45	188.77
	5	0.826	3	31.39	0.44	35.3	120	18.7	169.14	0	183.15
	Avg	**0.826**	**3**	**43.574**	**0.564**	**46.778**	**120**	**18.7**	**239.26**	**0.674**	**254.334**
Sunflower seeds HO	1	2.4	16	21.85	0.4	24.37	32	4.9	93.28	0.46	99.65
	2	2.4	16	22.8	0.38	25.42	32	4.9	81.3	0.59	85.71
	3	2.4	16	24.67	0.36	26.61	32	4.9	90.73	0.61	94.8
	4	2.4	16	23.44	0.36	26.06	32	4.9	86.17	0.47	90.33
	5	2.4	16	24.03	0.36	25.98	32	4.9	90.32	0.58	94.5
	Avg	**2.4**	**16**	**23.358**	**0.372**	**25.688**	**32**	**4.9**	**88.36**	**0.542**	**92.998**
Mais kernels	1	1.2	7	43.58	0.38	46.64	161	23.6	193.45	0.63	197.68
	2	1.2	7	41.03	0.35	43.53	161	23.6	158.86	0.65	162.9
	3	1.2	7	56.84	0.5	59.75	161	23.6	152.36	0.53	156.45
	4	1.2	7	42.44	0.38	45.03	161	23.6	159.89	0.51	163.78
	5	1.2	7	38.59	0.55	41.41	161	23.6	145.54	0.6	149.93
	Avg	**1.2**	**7**	**44.496**	**0.432**	**47.272**	**161**	**23.6**	**162.02**	**0.584**	**166.148**
Barley	1	1	5	35.03	0.37	36.63	48	7.1	113.44	0.46	117.3
	2	1	5	40.22	0.43	41.56	48	7.1	101.49	0.47	105.79
	3	1	5	35.7	0.3	38.34	48	7.1	99.06	0.62	103.45
	4	1	5	36.47	0.36	38.12	48	7.1	124.15	0.59	129.19
	5	1	5	36.66	0.27	38.06	48	7.1	93.88	0.53	98.77
	Avg	**1**	**5**	**36.816**	**0.346**	**38.542**	**48**	**7.1**	**106.4**	**0.534**	**110.9**

Table A.7 - Loading time part 2 - Table and Canton Data

Source		**Table**					**Cantons**				
	Nr.	Count	Size [kB]	Waiting [ms]	Down -load [ms]	Time total [ms]	Count	Size [kB]	Waiting [ms]	Down-load [ms]	Time total [ms]
Corn silage survey	1	3956	16.5	3750	0.53	3770	22	5900	1380	423.39	1810
	2	3956	16.5	4030	0.55	4040	22	5900	1360	464.81	1830
	3	3956	16.5	3720	0.62	3730	22	5900	1480	489.45	1970
	4	3956	16.5	3650	0.78	3650	22	5900	1340	517.09	1860
	5	3956	16.5	3710	0.73	3720	22	5900	1340	483.75	1830
	Avg	**3956**	**16.5**	**3772**	**0.642**	**3782**	**22**	**5900**	**1380**	**475.698**	**1860**
Hay survey 2018	1	2354	14.6	3.42	0.67	3430	23	5800	2030	459.79	2500
	2	2354	14.6	3380	0.57	3390	23	5800	1970	463.87	2440
	3	2354	14.6	3240	0.45	3240	23	5800	1950	442.96	2400
	4	2354	14.6	3240	0.43	3250	23	5800	1920	472.64	2410
	5	2354	14.6	3390	0.52	3400	23	5800	1930	469.87	2410

	Avg	**2354**	**14.6**	**2650.68**	**0.528**	**3342**	**23**	**5800**	**1960**	**461.826**	**2432**
Mountain hay > 1000 m, 2005-2017	1	2679	14.8	3430	0.64	3430	19	5000	1900	392.1	2300
	2	2679	14.8	3480	0.72	3490	19	5000	1930	405.73	2340
	3	2679	14.8	3440	0.63	3450	19	5000	1890	396.35	2300
	4	2679	14.8	3400	0.63	3400	19	5000	1840	378.36	2220
	5	2679	14.8	3390	0.57	3400	19	5000	1840	393.35	2240
	Avg	**2679**	**14.8**	**3428**	**0.638**	**3434**	**19**	**5000**	**1880**	**393.178**	**2280**
Hay 1st cut: sugar regional pattern and correlation with ADF	1	4610	11.2	2460	0.39	2460	25	6300	1350	521.99	1870
	2	4610	11.2	2730	0.48	2730	25	6300	1260	475.62	1740
	3	4610	11.2	2630	0.4	2630	25	6300	1410	505.71	1920
	4	4610	11.2	2620	0.61	2620	25	6300	1260	483.35	1750
	5	4610	11.2	2620	0.38	2620	25	6300	1250	501.67	1760
	Avg	**4610**	**11.2**	**2612**	**0.452**	**2612**	**25**	**6300**	**1306**	**497.668**	**1808**
Hay survey regional NEL content 2005-2017	1	16326	10.2	2500	0.62	2510	24	6100	1370	466.81	1840
	2	16326	10.2	2430	0.49	2430	24	6100	1300	464.63	1780
	3	16326	10.2	2460	0.56	2470	24	6100	1330	497.6	1830
	4	16326	10.2	2450	0.34	2460	24	6100	1310	461.79	1780
	5	16326	10.2	2460	0.68	2470	24	6100	1280	468.68	1750
	Avg	**16326**	**10.2**	**2460**	**0.538**	**2468**	**24**	**6100**	**1318**	**471.902**	**1796**
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	1	305	12.3	705.13	1.35	710.32	7	2500	933.12	203.38	1150
	2	305	12.3	376.93	1.48	392.78	7	2500	748.96	226.08	991.08
	3	305	12.3	686.53	1.44	709.99	7	2500	669.43	232.52	950.46
	4	305	12.3	424.61	1.37	437.66	7	2500	809.55	230.52	1050
	5	305	12.3	500.77	1.6	514.53	7	2500	1060	229.43	1300
	Avg	**305**	**12.3**	**538.794**	**1.448**	**553.056**	**7**	**2500**	**844.212**	**224.386**	**1088.308**
Sunflower seeds HO	1	36	16.7	295.38	0.41	300.34	1	483	194.82	37.53	236.58
	2	36	16.7	298.9	0.41	303.27	1	483	186.04	34.78	225.13
	3	36	16.7	297.52	0.42	301.74	1	483	201.9	34.95	240.88
	4	36	16.7	314.88	0.51	319.75	1	483	192.83	34.26	231.26
	5	36	16.7	317.18	0.4	321.39	1	483	198.15	33.23	235.49
	Avg	**36**	**16.7**	**304.772**	**0.43**	**309.298**	**1**	**483**	**194.748**	**34.95**	**233.868**
Mais kernels	1	2152	12.2	1740	0.46	1750	9	3800	814.96	281.26	1100
	2	2152	12.2	1820	0.58	1820	9	3800	742.2	306.27	1050
	3	2152	12.2	1720	0.7	1720	9	3800	756.91	298.27	1060
	4	2152	12.2	1740	0.41	1750	9	3800	732.93	304.92	1040
	5	2152	12.2	1710	0.4	1710	9	3800	791.66	311.09	1110
	Avg	**2152**	**12.2**	**1746**	**0.51**	**1750**	**9**	**3800**	**767.732**	**300.362**	**1072**
Barley	1	1528	13.5	1540	0.48	1540	13	3600	685.92	278.58	968.62
	2	1528	13.5	1490	0.46	1500	13	3600	654.35	285.8	941.2
	3	1528	13.5	1470	0.41	1470	13	3600	660.01	276.94	944.32
	4	1528	13.5	1430	0.39	1440	13	3600	714.26	286.36	1003.59
	5	1528	13.5	1460	0.4	1460	13	3600	660.75	311.2	996.32
	Avg	**1528**	**13.5**	**1478**	**0.428**	**1482**	**13**	**3600**	**675.058**	**287.776**	**970.81**

Table A.8 - Loading time part 3 - Location and Density Data

Source		**Location**					**Density**				
	Nr.	Points	Size [kB]	Waiting [ms]	Down -load [ms]	Time total [ms]	Points	Size [kB]	Waiting [ms]	Down -load [ms]	Time total [ms]
Corn silage survey	1	743	90.1	957.89	0.62	962.74	731	46.3	165.05	0.49	168.69
	2	743	90.1	1070	0.64	1070	731	46.3	167.07	0.54	170.85
	3	743	90.1	926.1	0.62	934.69	731	46.3	154.7	0.45	158.23
	4	743	90.1	933.08	0.61	937.66	731	46.3	160.79	0.5	164.27
	5	743	90.1	981.27	0.62	990.27	731	46.3	198.72	0.48	202.26
	Avg	**743**	**90.1**	**973.668**	**0.622**	**979.072**	**731**	**46.3**	**169.266**	**0.492**	**172.86**
Hay survey 2018	1	636	77.3	1030	0.66	1040	627	39.5	360.35	0.38	363.7
	2	636	77.3	959.69	0.59	964.02	627	39.5	357.81	0.54	361.08
	3	636	77.3	870.01	0.59	873.97	627	39.5	351.89	0.4	355.4
	4	636	77.3	884.92	0.56	888.62	627	39.5	353.55	0.38	356.69
	5	636	77.3	828.39	0.6	833.88	627	39.5	354.85	0.44	357.8
	Avg	**636**	**77.3**	**914.602**	**0.6**	**920.098**	**627**	**39.5**	**355.69**	**0.428**	**358.934**
Mountain hay > 1000 m, 2005-2017	1	438	53.4	1100	0.63	1110	425	26.9	440.87	0.8	444.21
	2	438	53.4	1120	0.59	1120	425	26.9	361.36	1.19	365.23
	3	438	53.4	1190	0.6	1200	425	26.9	384.89	0.54	388.2
	4	438	53.4	1150	0.59	1150	425	26.9	423.96	0.4	426.76
	5	438	53.4	1230	0.59	1240	425	26.9	455.21	0.53	458.56
	Avg	**438**	**53.4**	**1158**	**0.6**	**1164**	**425**	**26.9**	**413.258**	**0.692**	**416.592**
Hay 1st cut: sugar regional pattern and correlation with ADF	1	978	119	1000	0.78	1010	924	58.1	87.29	0.53	91.02
	2	978	119	529.34	1.04	534.52	924	58.1	92.75	0.51	96.46
	3	978	119	499.59	0.82	504.13	924	58.1	126.06	0.49	129.3
	4	978	119	528.57	0.89	538.05	924	58.1	85.58	0.51	89.06
	5	978	119	491.67	0.96	496.28	924	58.1	130.11	0.52	133.73
	Avg	**978**	**119**	**609.834**	**0.898**	**616.596**	**924**	**58.1**	**104.358**	**0.512**	**107.914**
Hay survey regional NEL content 2005-2017	1	2271	276	551.04	2.08	561.13	2153	134	125.11	0.91	129.08
	2	2271	276	541.07	1.95	547.05	2153	134	112.62	0.74	116.19
	3	2271	276	360.45	2.61	367.11	2153	134	120.63	0.76	124.18
	4	2271	276	408.22	1.8	413.96	2153	134	123.24	0.72	126.88
	5	2271	276	570.28	1.7	579.46	2153	134	159.94	1.08	163.99
	Avg	**2271**	**276**	**486.212**	**2.028**	**493.742**	**2153**	**134**	**128.308**	**0.842**	**132.064**
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	1	11	1.3	102.69	1.08	114.33	10	0.627	18.43	0.46	27.79
	2	11	1.3	63.43	0.95	75.45	10	0.627	37.31	0.44	47.32
	3	11	1.3	32.22	1.11	41.81	10	0.627	155.4	0.55	42.75
	4	11	1.3	74.23	0.8	81.16	10	0.627	27.03	0.46	46.6
	5	11	1.3	33.31	0.7	43.28	10	0.627	16.96	0.61	27.58
	Avg	**11**	**1.3**	**61.176**	**0.928**	**71.206**	**10**	**0.627**	**51.026**	**0.504**	**38.408**
Sunflower seeds HO	1	3	0.364	93.53	0.33	97.36	2	0.129	13.91	0.36	17.25
	2	3	0.364	91.03	0.37	95.02	2	0.129	14.15	0.4	17.54
	3	3	0.364	115.84	0.34	120.34	2	0.129	13.4	0.36	16.99
	4	3	0.364	88.54	0.33	91.62	2	0.129	144.84	0.52	18.72
	5	3	0.364	95.03	0.43	98.42	2	0.129	13.2	0.4	17.51
	Avg	**3**	**0.364**	**96.794**	**0.36**	**100.552**	**2**	**0.129**	**39.9**	**0.408**	**17.602**
Mais kernels	1	26	3.1	143.55	0.7	147.95	24	1.5	26.34	0.36	29.7
	2	26	3.1	135.93	0.41	140.08	24	1.5	26.02	0.47	29.57

	3	26	3.1	137.28	0.51	137.28	24	1.5	29.97	0.52	33.34
	4	26	3.1	143.72	0.49	148.43	24	1.5	25.63	0.38	29.04
	5	26	3.1	135.94	0.41	140.28	24	1.5	38.78	0.43	42.1
	Avg	**26**	**3.1**	**139.284**	**0.504**	**142.804**	**24**	**1.5**	**29.348**	**0.432**	**32.75**
Barley	1	73	8.8	62.14	0.42	66.26	70	4.4	21.11	0.41	23.93
	2	73	8.8	73.72	0.5	74	70	4.4	23.22	0.39	26.49
	3	73	8.8	74.51	0.44	74.75	70	4.4	39.44	0.51	42.75
	4	73	8.8	56.94	0.41	56.89	70	4.4	38.25	0.36	41.5
	5	73	8.8	60.8	0.63	61.11	70	4.4	28.73	0.37	31.54
	Avg	**73**	**8.8**	**65.622**	**0.48**	**66.602**	**70**	**4.4**	**30.15**	**0.408**	**33.242**

Table A.9 - Loading time part 4 - Regression Data

Source		**Regression**				
	Nr.	Points	Size [kB]	Waiting [ms]	Download [ms]	Time total [ms]
Corn silage survey	1	617	92.9	1190	0.75	1200
	2	617	92.9	1180	0.76	1180
	3	617	92.9	1170	0.84	1170
	4	617	92.9	1180	0.76	1180
	5	617	92.9	1200	0.77	1210
	Avg	**617**	**92.9**	**1184**	**0.776**	**1188**
Hay survey 2018	1	483	72.4	774.35	0.69	778.46
	2	483	72.4	757.4	0.67	761.12
	3	483	72.4	726.26	0.93	730.31
	4	483	72.4	737.7	0.83	741.89
	5	483	72.4	737.07	0.83	741.26
	Avg	**483**	**72.4**	**746.556**	**0.79**	**750.608**
Mountain hay > 1000 m, 2005-2017	1	332	50	392	0.59	395.59
	2	332	50	466.47	0.62	470.61
	3	332	50	471.77	0.59	475.68
	4	332	50	466.68	0.6	470.24
	5	332	50	472.41	0.6	476.3
	Avg	**332**	**50**	**453.866**	**0.6**	**457.684**
Hay 1st cut: sugar regional pattern and correlation with ADF	1	683	103	1550	0.75	1560
	2	683	103	1420	0.84	1420
	3	683	103	1460	0.82	1460
	4	683	103	1440	0.85	1450
	5	683	103	1420	0.88	1420
	Avg	**683**	**103**	**1458**	**0.828**	**1462**
Hay survey regional NEL content 2005-2017	1	2153	333	141450	5.7	14160
	2	2153	333	14090	5.81	14100
	3	2153	333	14180	5.68	14190
	4	2153	333	14020	5.93	14030
	5	2153	333	14140	5.99	14150
	Avg	**2153**	**333**	**39576**	**5.822**	**14126**
Correlation between CP and fat GC>2011 in	1	9	1.5	109.62	1.11	117.83
	2	9	1.5	194.36	0.77	198.98
	3	9	1.5	72.93	0.87	81.89
	4	9	1.5	72.77	1.13	89.81

rapeseed standard and HOLL	5	9	1.5	218.91	0.91	228.31
	Avg	**9**	**1.5**	**133.718**	**0.958**	**143.364**
Sunflower seeds HO	1	2	0.45	66.45	0.43	70.13
	2	2	0.45	86.88	0.68	90.76
	3	2	0.45	60.84	0.52	64.61
	4	2	0.45	63.1	0.52	66.83
	5	2	0.45	62.84	0.41	66.36
	Avg	**2**	**0.45**	**68.022**	**0.512**	**71.738**
Mais kernels	1	18	2.9	71.82	0.56	75.65
	2	18	2.9	67.78	0.56	72.47
	3	18	2.9	68.17	0.49	71.94
	4	18	2.9	67.25	0.63	70.98
	5	18	2.9	68.24	0.46	72.03
	Avg	**18**	**2.9**	**68.652**	**0.54**	**72.614**
Barley	1	21	3.4	118.79	0.46	122.6
	2	21	3.4	79.45	0.46	82.97
	3	21	3.4	68.79	0.38	72.57
	4	21	3.4	75.14	0.45	78.81
	5	21	3.4	79.75	0.57	83.98
	Avg	**21**	**3.4**	**84.384**	**0.464**	**88.186**

A.2.4 Initialization time

Table A.10 - Initialization time of the different visualization parts (bold = average)

Source	**Canton Map**		**Scatter Chart**		**Table**	
	Size (kB)	Init (ms)	Points	Init (ms)	Size (kB)	Init (ms)
Corn silage survey	5900	193.01	743	53.66	16.5	2.99
	5900	188.88	743	78.62	16.5	2.48
	5900	198.91	743	85.17	16.5	2.38
	5900	191.41	743	77.41	16.5	2.34
	5900	207.05	743	83.73	16.5	2.42
	5900	**195.852**	**743**	**75.718**	**16.5**	**2.522**
Hay survey 2018	5800	168.83	636	26.35	14.6	2.5
	5800	197.86	636	28.32	14.6	2.5
	5800	200.71	636	28.21	14.6	2.54
	5800	161.35	636	28.71	14.6	2.53
	5800	179.92	636	29.1	14.6	2.51
	5800	**181.734**	**636**	**28.138**	**14.6**	**2.516**
Mountain hay > 1000 m, 2005-2017	5000	183.58	438	45.72	14.8	3.22
	5000	177.51	438	41.02	14.8	2.65
	5000	174.51	438	42	14.8	2.66
	5000	171.72	438	41.75	14.8	3.15
	5000	182.88	438	45.25	14.8	3.11
	5000	**178.04**	**438**	**43.148**	**14.8**	**2.958**
Hay 1st cut: sugar regional pattern and correlation with ADF	6300	144.65	980	77.69	11.2	1.91
	6300	178.54	980	76.86	11.2	1.89
	6300	159.94	980	78.04	11.2	1.93
	6300	138.09	980	54.48	11.2	1.88
	6300	135.91	980	79.98	11.2	1.94
	6300	**151.426**	**980**	**73.41**	**11.2**	**1.91**
	6100	194.66	2271	76.11	10.2	1.76

Hay survey regional NEL content 2005-2017	6100	183.53	2271	68.21	10.2	1.8
	6100	171.56	2271	70.18	10.2	1.79
	6100	178.83	2271	70.75	10.2	1.77
	6100	174.65	2271	71.93	10.2	1.75
	6100	**180.646**	**2271**	**71.436**	**10.2**	**1.774**
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	2500	75.31	120	35.51	12.3	1.84
	2500	73.81	120	32.77	12.3	1.87
	2500	78.07	120	33.43	12.3	1.85
	2500	75.71	120	34.14	12.3	1.86
	2500	75.28	120	31.7	12.3	1.96
	2500	**75.636**	**120**	**33.51**	**12.3**	**1.876**
Sunflower seeds HO	483	38.5	3	28.39	16.7	2.36
	483	35.95	3	33.37	16.7	2.53
	483	40.26	3	28.73	16.7	2.25
	483	36.38	3	32.02	16.7	2.42
	483	42.97	3	35.21	16.7	4.04
	483	**38.812**	**3**	**31.544**	**16.7**	**2.72**
Mais kernels	3800	139.37	161	26.61	12.2	2.01
	3800	100.8	161	44.97	12.2	2
	3800	167.45	161	25.4	12.2	2.02
	3800	129.09	161	25.56	12.2	1.98
	3800	142.21	161	25.6	12.2	2.06
	3800	**135.784**	**161**	**29.628**	**12.2**	**2.014**
Barley	3600	154.63	73	26.9	13.5	1.92
	3600	137.53	73	24.23	13.5	1.89
	3600	99.72	73	27.11	13.5	1.93
	3600	95.98	73	29.81	13.5	2.24
	3600	111.67	73	37.62	13.5	1.92
	3600	**119.906**	**73**	**29.134**	**13.5**	**1.98**

Table A.11 - Detailed initialization time for scatter chart (bold = average)

Source	Size [kB]	Init map [ms]	Zoom listener [ms]	Overlay adding [ms]	Location listener [ms]	Coloration [ms]
Corn silage survey	5900	7.56	0.02	180.63	0.16	1.18
	5900	8.11	0.02	178.95	0.1	0.71
	5900	8.31	0.04	151.61	0.1	0.69
	5900	7.2	0.03	192.85	0.13	0.8
	5900	8.53	0.04	192.5	0.11	0.69
	5900	**7.942**	**0.03**	**179.308**	**0.12**	**0.814**
Hay survey 2018	5800	8.58	0.03	161.63	0.111	0.73
	5800	8.96	0.05	165.62	0.1	0.74
	5800	10.24	0.04	144.19	0.15	0.87
	5800	8.77	0.04	195.8	0.14	0.73
	5800	9.04	0.05	178.87	0.15	0.79
	5800	**9.118**	**0.042**	**169.222**	**0.1302**	**0.772**
Mountain hay > 1000 m, 2005-2017	5000	10.14	0.03	175.05	0.12	0.7
	5000	7.63	0.02	172.69	0.14	0.82
	5000	9.13	0.04	166.48	0.11	0.71
	5000	8.82	0.03	166.15	0.13	0.82
	5000	8.72	0.03	162.32	0.14	0.88
	5000	**8.888**	**0.03**	**168.538**	**0.128**	**0.786**
Hay 1st cut: sugar regional pattern and correlation with ADF	6300	6.39	0.02	163.57	0.13	0.74
	6300	9.58	0.05	157.71	0.12	0.74
	6300	7.15	0.03	223.67	0.12	0.77
	6300	7.44	0.03	130.65	0.12	0.81
	6300	9.98	0.02	131.57	0.13	0.87
	6300	**8.108**	**0.03**	**161.434**	**0.124**	**0.786**

Hay survey regional NEL content 2005-2017	6100	7.53	0.02	131.14	0.14	0.8
	6100	8.62	0.02	184.83	0.14	0.8
	6100	7.25	0.02	168.92	0.1	0.67
	6100	8.77	0.04	165.46	0.14	0.81
	6100	7.12	0.02	184.21	0.19	1
	6100	**7.858**	**0.024**	**166.912**	**0.142**	**0.816**
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	2500	9.64	0.02	107.21	0.13	0.83
	2500	11.11	0.02	66.7	0.21	1.08
	2500	8.83	0.02	64.69	0.13	0.81
	2500	9.93	0.02	66.88	0.13	1.07
	2500	9.29	0.03	66.37	0.13	0.77
	2500	**9.76**	**0.022**	**74.37**	**0.146**	**0.912**
Sunflower seeds HO	483	9.88	0.02	24.95	0.16	0.91
	483	11.09	0.02	27.09	0.14	1.25
	483	8.62	0.02	26.67	0.16	0.95
	483	7.24	0.02	29.95	0.17	1.03
	483	9.12	0.03	23.8	0.12	0.84
	483	**9.19**	**0.022**	**26.492**	**0.15**	**0.996**
Mais kernels	3800	12.55	0.03	143.32	0.14	0.79
	3800	7.02	0.03	126.5	0.13	0.8
	3800	7.08	0.02	89.73	0.13	0.85
	3800	9.83	0.02	138.49	0.13	0.79
	3800	7.96	0.03	128.34	0.16	0.86
	3800	**8.888**	**0.026**	**125.276**	**0.138**	**0.818**
Barley	3600	9.52	0.02	82.89	0.12	0.79
	3600	11.32	0.02	140.36	0.14	0.84
	3600	7.09	0.02	122.93	0.13	0.8
	3600	7.68	0.02	84.25	0.13	0.86
	3600	9.34	0.03	126.77	0.12	0.78
	3600	**8.99**	**0.022**	**111.44**	**0.128**	**0.814**

Table A.12 - Detailed initialization time for canton map (bold = average)

Source	Data points	Add Tag [ms]	Process geoJson [ms]	initChart [ms]	Init chart w/o events [ms]
Corn silage survey	743	0.2	0.97	82.42	66.14
	743	0.16	0.75	85.73	67.21
	743	0.18	0.85	79.88	67.28
	743	0.13	0.74	81.67	64.65
	743	0.17	0.85	85.5	68.07
	743	**0.168**	**0.832**	**83.04**	**66.67**
Hay survey 2018	636	0.16	0.35	27.11	23.63
	636	0.16	0.34	27.15	24.09
	636	0.18	0.41	42.06	22.21
	636	0.13	0.29	26.47	23.06
	636	0.15	0.35	26.69	23.61
	636	**0.156**	**0.348**	**29.896**	**23.32**
Mountain hay > 1000 m, 2005-2017	438	0.16	0.62	36.85	35.38
	438	0.16	0.69	37.26	30.44
	438	0.14	0.58	34.53	27.64
	438	0.16	0.63	38.13	30.27
	438	0.15	0.57	36.78	28.49
	438	**0.154**	**0.618**	**36.71**	**30.444**
Hay 1st cut: sugar regional pattern and correlation with ADF	980	0.17	0.85	77.38	71.17
	980	0.12	0.64	75.64	67.5
	980	0.12	0.65	75.52	70.23
	980	0.17	0.87	75.88	69.19
	980	0.2	0.94	74.25	71.13

	980	**0.156**	**0.79**	**75.734**	**69.844**
Hay survey regional NEL content 2005-2017	2271	0.17	0.76	49.55	43.14
	2271	0.16	0.76	49.25	43.41
	2271	0.16	0.75	49.08	42.21
	2271	0.17	0.75	49.21	43.16
	2271	0.17	0.75	48.98	42.2
	2271	**0.166**	**0.754**	**49.214**	**42.824**
Correlation between CP and fat GC>2011 in rapeseed standard and HOLL	120	0.2	0.32	31.31	28.17
	120	0.17	0.33	28.93	31.05
	120	0.16	0.31	30.85	28.43
	120	0.16	0.37	31.35	28.72
	120	0.17	0.36	30.56	28.56
	120	**0.172**	**0.338**	**30.6**	**28.986**
Sunflower seeds HO	3	0.16	0.2	24.65	23.73
	3	0.18	0.22	28	21.57
	3	0.16	0.21	29.38	22.44
	3	0.31	0.33	28.22	22.03
	3	0.32	0.27	31.13	21.77
	3	**0.226**	**0.246**	**28.276**	**22.308**
Mais kernels	161	0.17	0.58	27.04	24.15
	161	0.16	0.39	24.66	25.05
	161	0.16	0.31	33.34	22
	161	0.16	0.3	25.59	22.97
	161	0.17	0.33	27.18	22.09
	161	**0.164**	**0.382**	**27.562**	**23.252**
Barley	73	0.17	0.25	26.1	23.57
	73	0.16	0.26	27.26	23.03
	73	0.16	0.24	24.89	29.55
	73	0.16	0.26	30.11	22.26
	73	0.17	0.25	24.05	21.33
	73	**0.164**	**0.252**	**26.482**	**23.948**