End Report
Project COAT

Henk Kant - *4024400*
Daniël Mast - *4092457*

June 30, 2013

Contents

Abstract

CHAIN*els* is a platform which enables companies to maintain their professional relationships. On the platform, companies can perform all kinds of actions, such as creating connections with other companies, post messages of different kinds, and define their supply. All of these actions create data which are stored in the CHAIN*els* database.

With the continuing growth of the platform it becomes increasingly difficult to get insights about the activities performed by the platform's users. As less personal contact exists between the companies and the CHAIN*els* team, it also becomes increasingly difficult to assess the effectiveness of the marketing efforts of the team. For these purposes CHAIN*els* requested a tool to gather analytical information about the usage of the platform.

We created the Online Administration Tool (OAT). This tool extracts information from the data produces by the platform, and analyses it to provide useful knowledge to the marketing and development team of CHAIN*els*. This information is shown to the users of the tool by charts created with the JavaScript library *d3.js*, in a way which is clear and easy to comprehend. As not all users should have access to all parts of the system, access rights management is a core functionality of the OAT. A plugin-system ensures the system is easily extendable with new information from new features on the CHAIN*els* platform.

Testing the system has shown it is able to handle the information from the current CHAIN*els* platform, with enough room to let the number of signed up companies grow at least 10 times.

1 Introduction

In 2011 then-students Erwin Buckers and Vincent Koeman performed their bachelor's project. During this project, under the supervision of dr. Shahid Suddle, they went from an idea to a working platform called CHAIN*els*. After the launch in November 2011 it appeared the product was insufficient interesting to their target audience. The decision was made to do a grand makeover and relaunch the platform in 2012. During this second coming, businesses showed a greater interest and the first clients to sign up to the platform were a fact.

CHAIN*els* is a business-to-business social media platform. Where other platforms such as LinkedIn focus on individuals and their relationships, CHAIN*els* focuses on companies (particularly small and medium enterprises (SMEs)) as a whole.

Over the last two years the CHAIN*els* team grew from two to nearly a dozen. Through efforts from the team throughout the same period, the number of signed up companies has grown to a few hundred, and is expected to grow even further and faster in the near future. Most of the actions companies can do on the site generate data, and along with information about page-views, this data is stored in a database.

Some basic information was already extracted from this data, and some actions could be performed. However, to ensure the continuing growth of the platform, more information was required. Information regarding the use of the platform by the companies, and information regarding the effectiveness of the marketing efforts of the team. For this purpose, CHAIN*els* required a more extensive management tool.

For our project, we responded to this need by designing and creating the Online Administration Tool (OAT). This tool includes information extracted from the available data, the inclusion of the readily available management tools, rights management for the users, and a framework to enable easy expansion by the CHAIN*els* team when needed.

The structure of this document is as follows: in Section 2 we describe the problem at hand and the planning for this project. In Section 3 we outline the requirements, and in Section 4 the design of the new system. In Section 5 we discuss the actual implementation and testing. In Section 6 we discuss how we tested our implementation. In Section 7 we look back on the given requirements. In Section 8 we give recommendations and possible future improvements, before drawing a conclusion in Section 9.

2 Problem Analysis

2.1 The CHAIN*els* platform

The CHAIN*els* platform is based on connections between companies. A user is first required to sign up. His `account` is created at this point. At the same time, the user has to fill in the details of his company. Details include the company name, its mission, industry, address, from which they know CHAIN*els* and more. When that is done, the `company` is added to the system. Automatically, an `info page` is created, where the information of this company is presented (Figure 1). This page is publicly available to other users, not logged in visitors and search engines.

The next interesting step is forming a connection with another company, called a chain. When a user hits the button to suggest a connection with another company, a `chain request` is created. The user has to define what kind of chain it is: `employer/employee`, `relation` or `other`. When the other company accepts the request, the companies form a `chain` with each other. On the dynamically generated `chain page`, the existing chains of this company are presented (Figure 19 in Appendix A). If a user is happy with CHAIN*els* , he can send an `invite` email to someone else that does not already have a registered company on the platform. Invitations and chains are rewarded by providing `credits` to the company. Currently, these credits cannot be used yet, but will be in the future.

Companies can also post different types of messages. A `desk message` is used to spread news and tell chains what the company is up to. A `supply message` can be used to promote a product or workshop. An `event message` is used to announce an event. It is possible to write a `testimonial` on the info page of a chained company. Companies can `recommend` interesting messages to their chains, by hitting a button below the message. Also, chained companies can send `private messages` to each other, which can only be seen by these two companies.

A company can also add `employees`. Currently, these are only textually presented on the info page.

All above-mentioned entities are also classes in the back-end. Objects of these classes are stored in the database, which enables us to collect data of these entities. A screenshot of a company page is shown in Figure 1.

2.2 Current situation

Currently a basic tool exists in CHAIN*els*' system. This tool lets the team view some basic information, such as pageviews and invites to non-signed up companies, and perform some basic actions, such as deleting accounts. However, there is a lot more information available in the database. This information can be useful to get a greater insight in what the companies actually do on the platform, and can give a better foundation for future decisions on new functionality for the platform.

Amongst other things, CHAIN*els* is actively participating in conventions to spread the word about their platform. However, they can not easily assess whether those efforts translate to extra activity on the platform. Also, for companies which they do not have direct contact with, it is hard to determine the effectiveness of the marketing efforts.

2.3 Problem description

Our task was to create a system that fits CHAIN*els*' needs. This Online Administration Tool (OAT) needs to show any information of interest to the CHAIN*els* team. It needs to show this data in a way which is appropriate and easy to comprehend. It also needs to incorporate the data and functionality from the existing tool.

Multiple users will have access to the new tool. These users will not all have the same background, and so they need to use different aspects of the OAT. Some mechanics should be in place to determine which users can access which aspect.

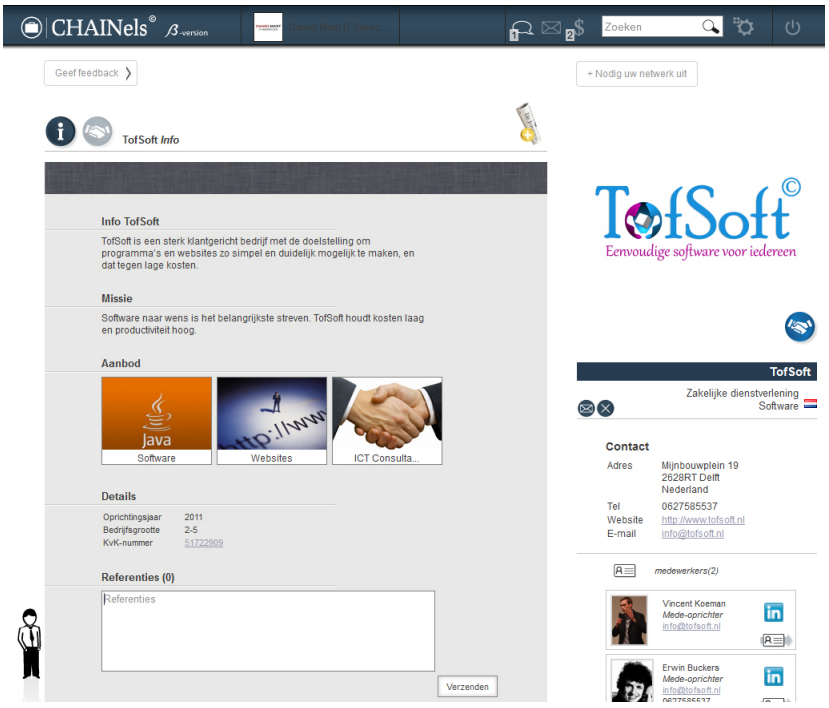

Figure 1: About-page of company TofSoft

The new system should be easily expandable in the future, to allow information gathering on new features of the platform. It should also be built in a way that enables the CHAIN*els* team to quickly understand and work with it.

For the problem description as provided by CHAIN*els*, see Appendix C: Plan of approach.

2.4 Planning

We divided this project into four phases. In the orientation phase, we established a plan of approach and did research into relevant topics. We researched two areas: Web Usage Mining, and efficiently displaying data. For our plan of approach, see Appendix C, and for the full Orientation Report, see Appendix D. In the second phase we assessed the requirements and created a design for our system. These will be detailed in Chapter 3, but for the full requirements document, see Appendix D. In the implementation phase we did the actual implementation on the new tool. In the final phase, we worked on finishing our product, this report, and the final presentation. Figure 2 shows a Gantt-chart displaying these phases. For a more extensive planning per phase, see Appendix B: Planning

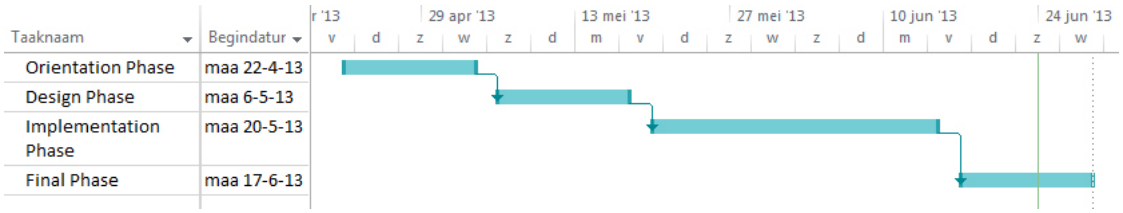

Figure 2: Planning of four phases

3 Requirements

This section discusses the requirements that were set up for the design of the system, before the implementation phase started. The requirements were set up in collaboration with the company.

3.1 Functional requirements

3.1.1 Analytics

The OAT should present analytical data, both in a visual and a textual format. The data should give insight in the current system of the CHAIN*els* website, and more specifically, into the activity of its users. Data about visitors, companies, messages and other elements should be provided. For the visual part, charts of different types should be created to present the data. Charts should provide options that filter the data to a certain time period, and to other categories, for example data about companies from a specified industry. The shown data should be up-to-date, without having the user to refresh the page manually. Below is a list of functionalities that the OAT should have:

- The total number of unique visitors, chains, new companies, credits, messages and invites in the system over a time period[1] to be specified by the user, or via predetermined time periods[2].
- The number of gained credits (total or per time period) should be dividable per action with which they were gained, for both real time and total numbers throughout the OAT.
- The registered companies should be dividable in industry, number of employees, and region, for both real time and total numbers throughout the OAT.
- The number of chains and invites should be dividable in sent requests and accepted requests throughout the OAT.
- For both chains and invites, the average time between requests and accepts should be available.
- For both chain requests and invites, a percentage of accepted requests should be calculated.
- The number of messages should be dividable per type of user[3] throughout the system.
- In real-time the number of online users (logged in and total) with logged in users dividable in region or industry. A real-time list of chain requests and accepts, invites sent and accepted, credits gained or spent, companies who joined, and messages sent, during a period that is customisable by the user.
- A list of companies, based on activity, to show most and least "active" users.
- The average amount of recommends and comments on posts. The amounts should be dividable in type of message[4] and in company number of employees, region, and industry throughout the OAT.
- How companies which signed up got to know CHAINels.
- Charts showing the percentages of different options in user/visitor statistics[5].

[1] No minimum or maximum time limits exist.
[2] 1 day, 1 week, 1 month.
[3] Currently only "Editor", "Manager" and "Spectator".
[4] Current types are "desk", "event", "supply".
[5] As of writing consisting of browser, location, operation system, account and company id (if any), and referrer page.

Per company:

- The amount of time it took the company to complete the sign-up process.
- The sign-up phase the company is in, and the percentage of fields (name, address, what services the company can offer, etc.) the company has filled out. The time they took to come get to that phase since sign-up, and the total time.
- The time they spent in the introduction manual of the website (if any).
- The total amount of chains, chain requests, credits, sent invites, and posts of the company.
- The amount of chains, credits, sent invites, received invites, and messages of the company over a time period specified by the user[1], or via predetermined time periods[2].
- A rating to show how active the company is.
- The pages a company views and the messages a company posts should be dividable in types of user accounts
- The average amount of recommends and comments on a post.

Per page:

- A visitors flow, showing where the user came from and what the next page the visitor goes to is (if any).

Per user:

- A list of statistics about the user[5].
- The amount of pages the user visits per session and on average.

Visualisation requirements:

- The modules that show real-time data should by default be refreshed every minute. Users should be able to refresh this data manually, or set a specified interval, as long as this is in accordance with the resource requirements.
- Visualisations of the data should be zoomable where possible, to increase detail of visualisations on the fly.

3.1.2 Management

The OAT should contain a department that provides management functions for the current system. This involves a data presentation of a company from the database, and functions to make changes.

- Users should be able to delete accounts of companies.
- The amount of credits which are rewarded for certain actions should be changeable.
- A notification to all users can be sent out to notify all companies at once.
- Companies' requests to be featured on the homepage should be handled through the OAT.
- Feedback from the companies should be viewable.
- Database access should be available.
- Error reports from the site should be viewable.

- Right management should be in place, to determine which user can view which parts of the OAT. New users should be easily added or removed. Rights of users should be editable.
- Notifications should always be visible to inform users about important events, like an error report.

3.1.3 Personalisation and rights management

Only members of the CHAIN*els* team should be able to access the tool. The team consists of people that each fulfil a different task. Currently, there are a CEO, two IT specialists, two marketing specialists, a designer, three interns and even more people with other functions active. For most of them, the OAT will be a valuable addition to their productivity. However, these individuals require different functionality from the OAT. Also, not all users should be able to access the same modules. Therefore, the OAT needs to be able to distinguish users and their roles in the company, and determine which modules it shows.

3.2 Nonfunctional requirements

3.2.1 Fit in existing framework

The current system of CHAIN*els* is a result of over two years development. Therefore, it was very convenient for the company if the OAT would use the same implementation style. The IT specialists would not have to learn a new system, and recognise the classes and methods that are utilised. Also, a lot of work would be saved, as the current system already contains lots of functions, for example the functions for communication with the database.

3.2.2 Modularity

One of the main desires of the company was being able to easily add modules by themselves at any time. A module is a small block containing a small portion of information, that is easily added to the system and made visible in the OAT, without having to change the code of the system itself. The website of CHAIN*els* is a continually evolving system. Therefore, it is very likely that our OAT has to be expanded as well, in order to enable analysis and management of the new features. Without modularity, the OAT would be outdated as soon as CHAIN*els* adds new functionality to their system.

3.2.3 Speed and ease of use

Users of the OAT should not have to wait a long time before the system has processed their request. This seems trivial, but can be a challenge, as an analytics tool has to store, retrieve and process great amounts of data. Moreover, usage of the tool should not be more complicated than necessary. It should be intuitive to use, and not require reading an extensive manual first. Charts should be presented in a quickly readable way. It should also not matter which operating system a user uses, and the tool should work on the latest version of all browsers.

4 Design

This section describes the design we made in the design phase. It encompasses the visual front-end design and the structural back-end design. As it was an initial design, features may be lacking. For those features, we describe the design as we made it at the point we needed to implement them. For the actual implementation, see Section 5.

As the OAT introduces new entities in the CHAIN*els* environment, we would like to give a short description:

Module A small section visible on all pages in the OAT. A module contains information as a chart or as plain text. All information in the OAT is placed in a module. All modules have unique ids, which are used to identify all modules in the system, and as the identifier in the database.

Chart A Chart represents a graphical line chart or bar chart.

Page A Page represents a regular web page. This contains the list of modules that are on that page.

My Page All pages in the OAT contain a predefined set of modules. However, when a user would like to combine modules from different pages, he can place them on their own personal My Page. My Page is a page customisable per user, and its contents and layout are stored in the database for each user.

AdminAccount Each user of the OAT has an AdminAccount. This account defines the user, and may contain certain attributes such as the rights the user has and the contents of his My Page.

Also, for this section another definition is needed:

component A black box which provides some functionality, without explicitly determining its implementation.

4.1 Data Design

The database CHAIN*els* uses is a Redis database. Redis is a so-called NoSQL database, which does not contain its data in schemas, but stores it in a key→value manner. When searching for a certain key in the database, the value of that key would be returned. For more information on Redis, see Appendix D: Orientation Report. As said, almost all actions which can be performed on the CHAIN*els* site create an object in the database. These objects all have a unique id. The objects are stored per type (company, chain, message etc.) in sets of key→value pairs, where the key is the id of the object, and the object itself is the value. If the object happens to have attributes, the value represents a key→value list of these attributes.

Initially, as it was a requirement to have the data as up to date as possible as long as the system was responsive, the required information was collected when needed. This way no calculated data was required in the database, and the only data design required was the already existent CHAIN*els* database. Although sufficiently fast when processing most type of objects, which run in the hundreds, it was not fast enough when collecting the objects for pageviews, which run in tens of thousands.

It was clear calculation had to be done beforehand. As mentioned in the requirements, the user should be able to see the data of a certain type of object on a daily, weekly, or monthly basis. For all types of objects which would be represented in a chart in the OAT, the necessary information would therefore be calculated per day and stored in the database. This calculated data would exist as an array with a certain date as a key and the corresponding information as the value belonging to that key. When storing this data, the array would be converted to a string and be assigned a string identifying the data as the key. All these keys were to be stored in a

single set, so that all data regarding the OAT would be grouped together in the database. See Figure 3 for a visual representation of this construction.

As each id in the CHAIN*els* database represents a unique object on their platform, when grouping per week or per month, the data per day could just be summed on a back-end level. An exception was the information about unique visitors. While grouping unique visitors per day sufficed for data per day, they also had to be calculated per week and per month, as some visitor one day may revisit the website on the next, and they would be counted double when just grouping days.

A module in the OAT is a new object in the CHAIN*els* system, and would stored as such in the database. The modules would have their own set next to the existing sets of objects. The unique identifying name of the module would be the key in the set, and as a module has attributes, a list of these attributes would be the value. As an AdminAccount and a Page would also be new objects in the system, they would also be stored in a set in the database, again with the id of the object as the key and the list with the attributes as the value. Again, see Figure 3 for a visual representation of this construction.

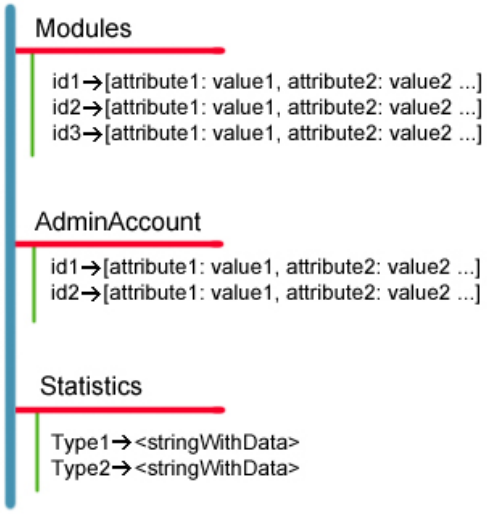

Figure 3: The model of the database. The blue vertical line represents the database. The horizontal red lines represent sets in the database, with their name above the line. Alongside the green vertical line is the content of the set. For Modules and AdminAccount these are are key→value pairs, with an id as the key, and a list of attributes as the value. For Statistics, the pair is a string identifying the data and a string-representation of an array.

4.2 Functionality separation

As our project was to manipulate sets of data, and then represent it to the user, we took this separation of different aspects and designed them accordingly. However, we did not look specifically at design patterns such as Model View Controller. Instead, we adhered to CHAIN*els*' way of separation of classes and packages. Specifically, we identified three parts in the new system which needed to be designed: the access to the data, the back-end logic where the system logic would be, and the front-end which would be visible to the user. This gives a structure of the system as shown in Figure 4. The Database Access part would retrieve and store any data that is needed by the OAT. The back-end logic would manipulate the data, create all elements, and perform all calculations needed. Actions from the user which required extra elements to be created, or required data from the database, are also part of this back-end. Elements which are required to create the visual interface would be passed to the front-end to display. The front-end part would then consist of displaying everything on screen, and handling all actions from a user that does not concern the back-end.

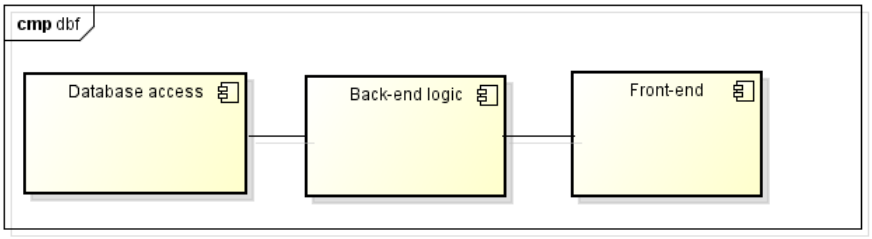

Figure 4: Separate elements in the system

4.2.1 Database access

For all types of objects in the database, a mapper exists in the CHAIN*els* system. These mappers can convert objects from the system to keys and values in the database, and can search for an id in the sets in the database and create the right object. For some objects, the corresponding mapper has some additional methods. To retrieve any information of the objects in the database, a mapper must be used. So, to create Page, AdminAccount, and Module objects, a mapper for these objects was needed.

With the need to calculate statistics, a new component was required with access to the database. This component would get all data, calculate whatever was needed, and store this in the 'Statistics' set in the database. This component therefore needed access to the mappers, and the ability to read from, and write to the database. Also, it was decided the number of components with direct access to the database should be minimised. All access to stored data would go through the component just described.

Now we can update the Database Access component with these required components, as shown in Figure 5.

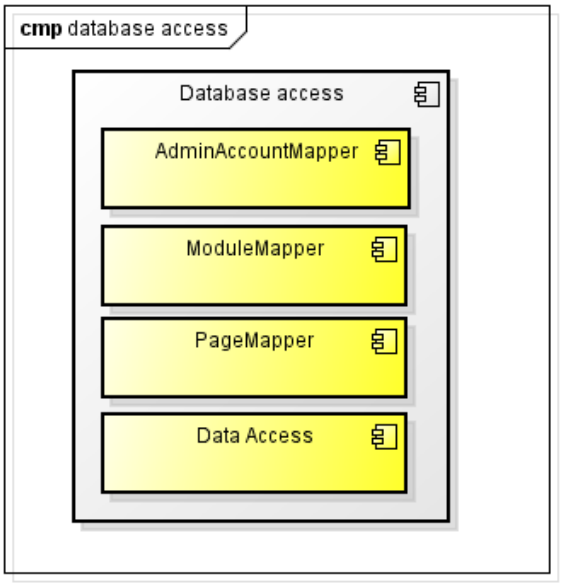

Figure 5: The designed Database Access part

4.2.2 Back-end

As it was a requirement the OAT should fit in the existing framework, the OAT system inherited the package structure of the CHAIN*els* system. This helped to increase familiarity of the CHAIN*els*

team with the OAT system. It was clear all files for the OAT system would be in a separate section, so it would be easily maintainable and intervention between the OAT and the CHAIN*els* system would be minimal. Some classes were already available, such as controller classes and the basis for the HTML pages.

Modules would be used to display all data on the screen. These would be divided into three types: ChartModules, TextModules, and ManagementModules. The first two types were to display data in a chart and in plain text, respectively, and the third would be used to display management features. Attributes of a Module would be the width of a module (single or double width), the rights that are required to see the module, and for ChartModules the type of chart (line chart or bar chart). Each module would be placed on at least one web page.

A web page in the system would be represented by the Page component. A Page would have attributes such as a list of the module ids that are on that specific page, and perhaps the right that is required to view that page. When viewing the web page, the Page object would be retrieved from the database and the Module objects would be created from the list of module ids from the Page.

An AdminAccount would be used to identify a user of the OAT. It would be required to access the tool, and have attributes such as the rights the user has in the system, and his My Page.

To handle all user input from the interface, actioncontrollers were required. They would be called on input that required data from the server, and they would return the response from the server. For example, when a user navigates from one page in the OAT to another, actioncontrollers would be responsible for restricting the refreshing of the page to only those parts that would actually be different. Also, when a user visits the CHAIN*els* site, the CHAIN*els* system uses a controller to determine which page the user should see. A similar controller was required for the OAT system.

For the back-end logic part as designed, see Figure 6.

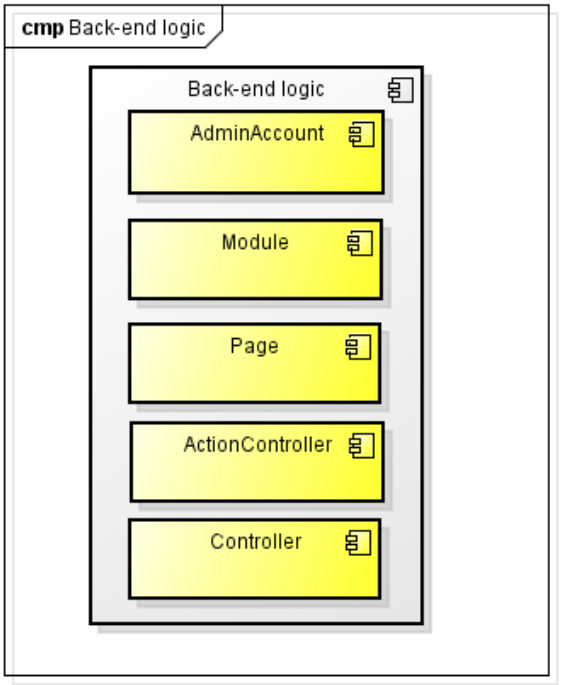

Figure 6: The designed Back-end logic part

4.2.3 Front-end

The front-end should contain three elements: a banner with a live feed, some method of navigation, and the body of the page. The live feed would show any changes in the system, such as newly signed-up companies or posted messages from that day. The body would contain all the information, that is, all the modules, of the page. The live feed was designed to update regularly, so it would always show fresh data.

The JavaScript library *d3.js* was to be used to create the charts in the ChartModules. When placing a module on a page, the necessary data would be retrieved from the database using an actioncontroller in the back-end, and passed on to the JavaScript function. Here certain aspects of the chart were calculated and the chart would be outputted. For up-to-date charts, on a set interval the script which displayed the chart would make an AJAX call to the server and retrieve the new data to update the chart.

One of the features of the modules would be that they would be draggable across the screen. By grabbing a predefined piece of the module, the user could rearrange the positions of the modules. On the user's My Page the layout could be stored at the account of the user.

For the front-end logic part as designed, see Figure 7.

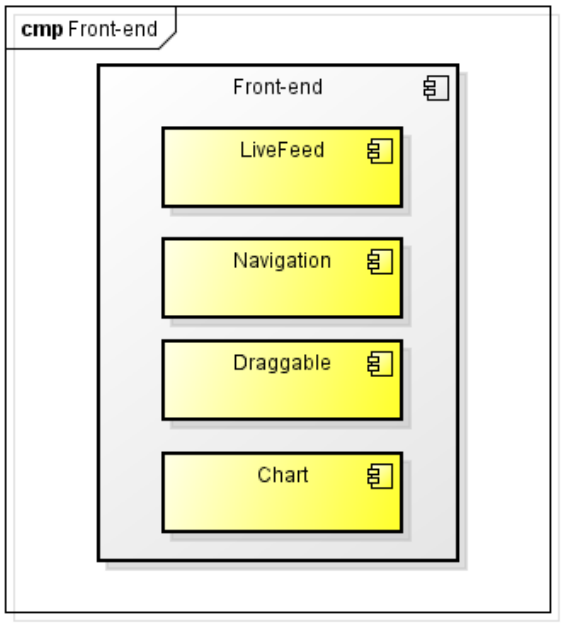

Figure 7: The designed Front-end part

4.3 Mockup

The interface was split up in three parts. At the top was a banner with live stats on the right. These live stats would display what was happening on the CHAIN*els* platform at that time. These statistics would update more frequently than any other statistics. The middle section contained navigation. The buttons for navigation would be grouped together to make a clear distinction between the type of page it leads to. Any page that would not be accessible to the user would be clearly marked so (like the 'Management' button in Figure 8). The lower part would contain all the modules. These modules would be single or double width. Different chart types would be possible, depending on the data to be displayed. Where possible the colour scheme of the CHAIN*els* website would be used, but the results of the orientation report were also taken into consideration.

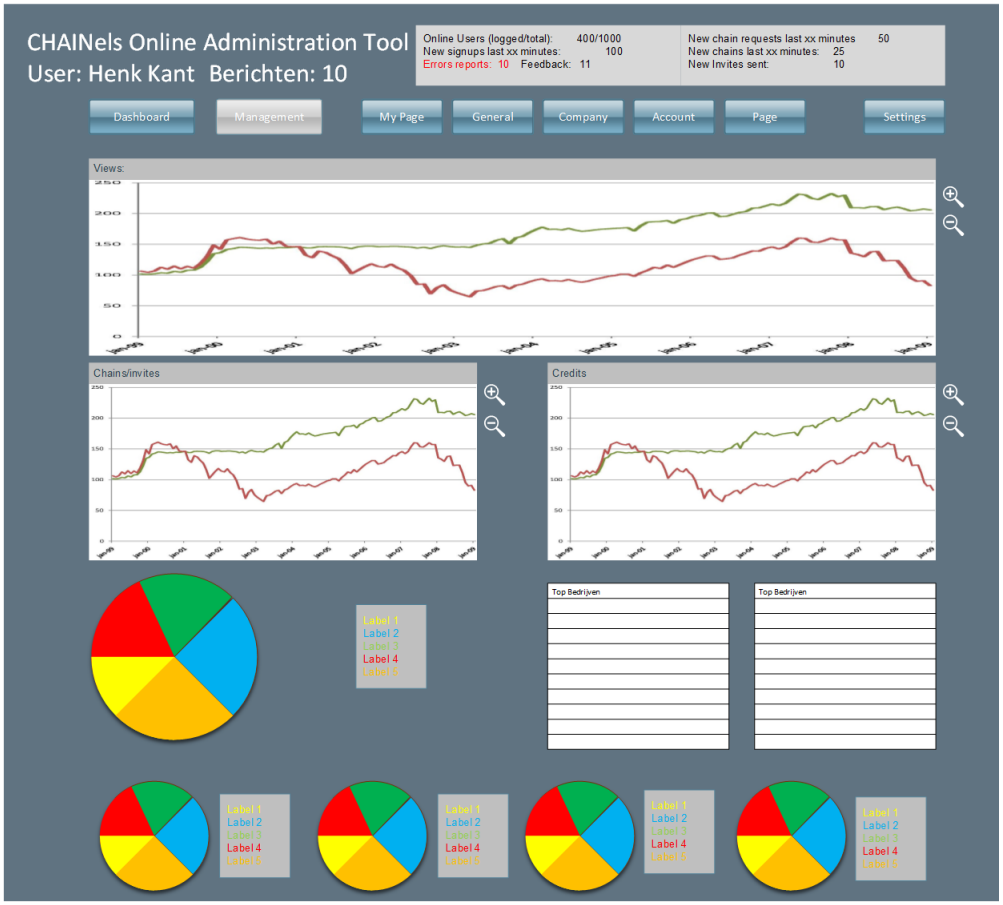

Figure 8: Mockup of the interface

The mock-up in Figure 8 shows a page in the OAT. On this page some linecharts and piecharts are displayed, as well as two charts with just text (lists in this example). Next to the linecharts are buttons to zoom those charts. In the navigation the 'Management' button is greyed out, indicating this user does not have the rights required to view this page. The banner shows a logo and the logged in user on the left, on the right is the live feed visible. This shows data such as currently logged in users on the CHAIN*els* platform and the number of companies which registered today.

5 Implementation

This section discusses the methods we used for creating the system, and the way in which we implemented it. In the end, it is explained how the tool was tested.

5.1 Architectural view

The class diagram of the OAT is presented in Figure 9. The different packages and classes are shown, with the relevant properties, methods and relations. The classes from the `statistics` and `module` packages have not been filled in detail in this diagram. They will be discussed separately in this chapter, as they form a major element in the system.

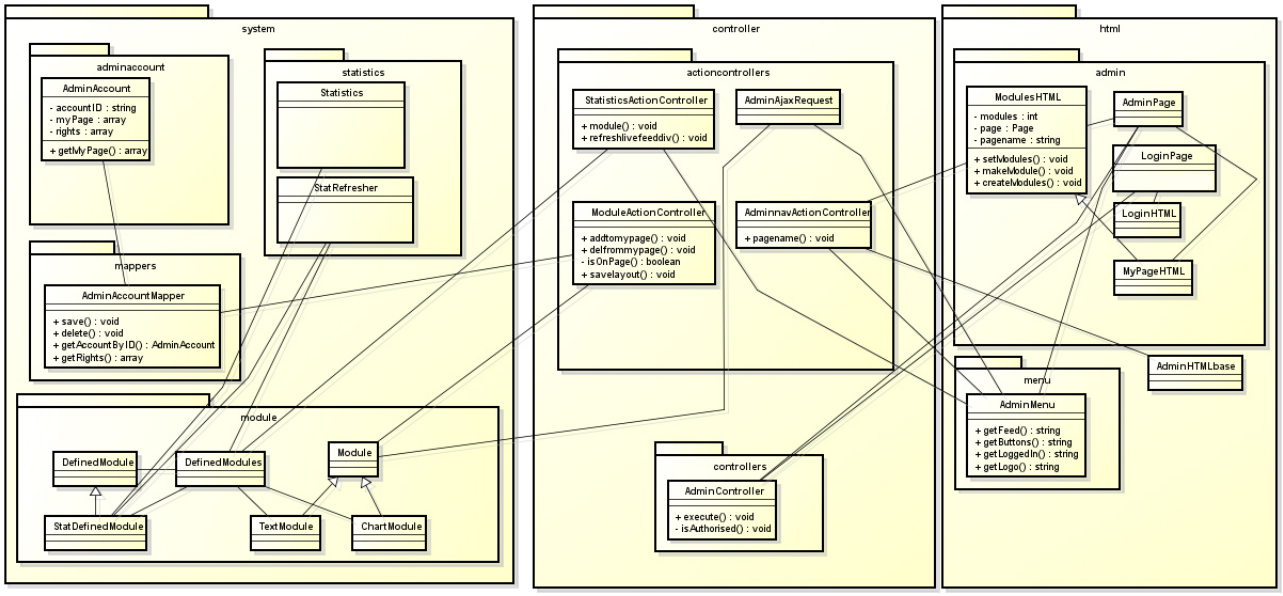

Figure 9: The class diagram after implementation

As can be seen in Figure 9, most components exist as designed. Also, a subdivision as in Model, View and Controller (MVC) becomes visible. In our system, this is named as `system`, `controller` and `html`. This is in conformity with the existing CHAIN*els* system.

5.2 Data Analysis and Aggregation

5.2.1 Storage and processing

An important decision we had to make was to what extent the statistics data should already be processed before storing. The advantage to do a lot of processing beforehand is that it takes less time to present the data to the user. Also, it requires less storage data, because only the data required for direct presentation is stored. This is also a disadvantage, as no processing can be done anymore when the raw data is not available anymore.

To satisfy the requirements, we had to hand in some speed in order to create more functionality. For all objects in the database the data is still grouped per day, but a list of company ids is stored for each day. Where there were multiple types for an object, such as different kinds of chains, the company ids are stored per day per type. By storing the company ids, a lot of information on the objects remains available. For instance, when viewing a chart with the newly created chains per day, the option is now available to only see chains created by certain companies.

The information is stored this way per day, as this is the minimal time period which a user can request. Each module stores its own data. This way the module id can be linked to the key

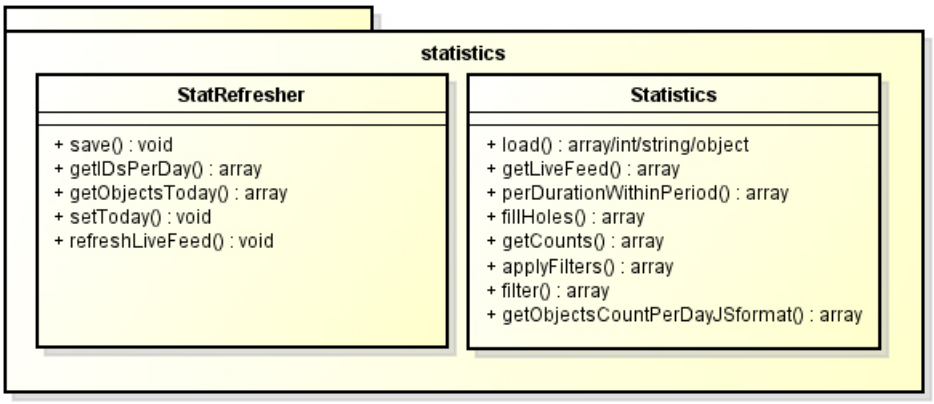

Figure 10: The `statistics` package, with classes `StatRefresher` and `Statistics`

in the database where the data is stored. If multiple modules need the same data, the storing of this data can be done in one module, and the other module can access it when necessary.

5.2.2 The `statistics` package

The part of the code in which modules differ is stored in the `setStatsToSave()` method of the module itself. However, there is also some functionality that is useful for more than one module. These functions are stored in the `statistics` package, in the classes `StatRefresher` and `Statistics`. See Figure 10. Here, the `save()` and `load()` functions make the actual database calls. For saving and loading statistics data, only these two functions are used, which makes it easy to change or add things in the future, and maintain uniformity.

The functions `getDurationWithinPeriod()`, `fillHoles()`, `getCounts()`, `applyFilters()`, `filter()` and `getObjectsCountPerDayJSformat()` are used by most modules that show charts. All functions require an array as input, and return an array. In this way, an array of data can be passed through, and is adjusted a bit in every function, passed to another function, and eventually is suitable for passing it to a JavaScript chart. Remember that the array that is passed through always contains a date as key, and an array of company ids as value.

- `getDurationWithinPeriod()`. Returns only the data that falls between a given begin and end date. Also, the data is bundled in a duration (day, week or month). This means that for a week or month, the data of days have to be summed up together.
- `fillHoles()`. Loops through the array from begin to end date and sets an empty array at dates that are not set at all. This is necessary to let the chart line go to 0 on that day.
- `applyFilters()` and `filter()`. Goes through all company ids in the array and removes all companies that do not satisfy the given categories.
- `getCounts()`. Eventually counts the number of company ids for a date, and returns that number, which will eventually be readable from the chart.
- `getObjectsCountPerDayJSformat()`. The last function, that transforms the way the array is built to a type that is required for passing it through to *d3.js*.

5.2.3 Updating the data

In the implementation phase, we quickly discovered that we had to uncouple the data refreshment from the moment that a user accesses the OAT. Otherwise, this user would have to wait unacceptably long before any data could be shown. Two separate scripts are written that refresh the

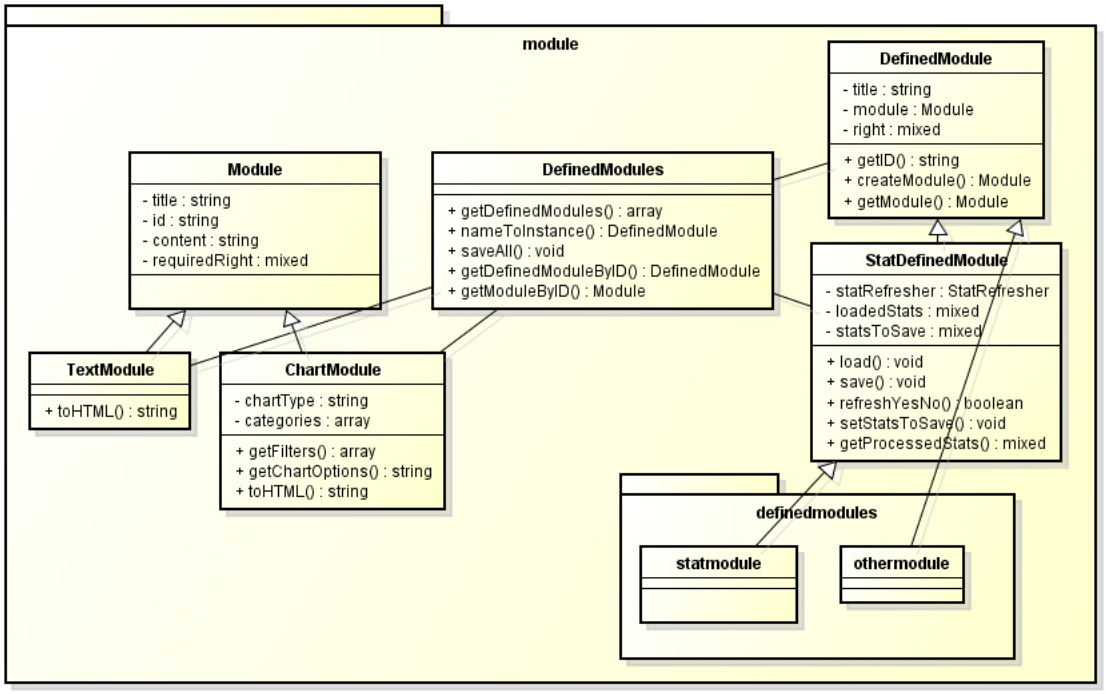

Figure 11: The `module` package

statistics data when run. One can be run to refresh the data for all modules. The second one only refreshes the live feed data. The second one takes less time to run, and has priority to be more up to date, and can therefore be run more often. These script are implemented in Cron, so that the the first script is run every hour, and the second one every minute.

5.2.4 Account details

Each user that is authorised to access the OAT, has his own AdminAccount. With this AdminAccount, information is stored about the user My Page. For each user, the ids of the modules that he has added to its My Page are storede. Also, information about the rights of the user is stored, and as these AdminAccount are linked to the CHAIN*els* account from the platform itself, a link has been made to that account.

5.3 Back-end

5.3.1 Modules

A Module is a small box containing some information about the CHAIN*els* system. These boxes have a size restriction, but this restriction ensures never too much is thrown at the user at once. Modules can have a normal width or the double width. Two types of Modules exist, `ChartModules` and `TextModules`. `ChartModules` always contain a chart. A chart is a graphic to display the data in a user friendly way, usually about the objects in the CHAIN*els* system. `TextModules` show their data in plain text. This can again be about the objects in the system, but also be management features. Each Module has a handle by which the user can move the Module across the page.

The class diagram of the `module` package is shown in Figure 11. When creating a new Module in the OAT, four steps have to be taken. These steps are shown in Figure 12.

First, its class should be created. The new class should either extend `DefinedModule` or its subclass `StatDefinedModule`, for a module with only text or with a chart, respectively. The created module class is put in the folder `definedmodules`. Figure 13 shows a simplified example

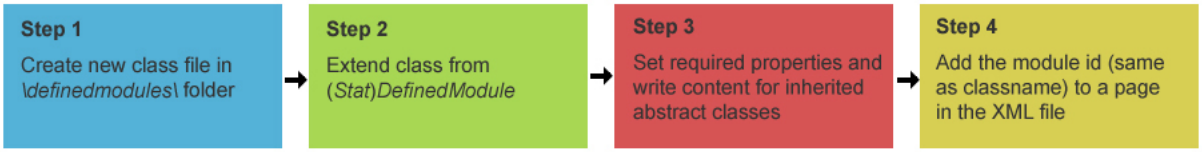

Figure 12: 4 Steps for adding a module

of a new `StatDefinedModule`. It shows which properties and methods should be set and what they should contain.

The final step is adding the created module to a page. For this, an XML file is used that lists the pages. For each page, the name of that page and the list of modules of that page are listed. Adding a module to a page is not more difficult than necessary, as can be seen in Figure 14. The id inside the `<module>` tag has to be the same as the classname of the new module. When these four are performed successfully, all the saving and loading of statistics data and the presentation of a module on a page is done automatically, and nowhere in the system extra details have to be set. A visual representation of adding a module, including the folder structure and XML file extension is shown in Figure 15.

Saving statistics

When refreshing all statistics, the folder `definedmodules` is read. A module is created from all classes in this folder. For modules that extend `StatDefinedModule` and should be refreshed (the variable `refresh` is not set to `false`) the method `setStatsToSave()` is run in the `saveAll()` method of `DefinedModules`. As shown in Figure 13, this method sets the statistics in a variable, and for all selected modules it is then stored in the database. The class names of the modules are obviously unique, and are therefore used as the id of the module. This id is then used as the key in the database where the statistics of the module are set. To store and process the statistics, the functions from the `statistics` package are used.

Loading the module

In Figure 16, a sequence diagram is shown from the moment when a user accesses a page. The `AdminController` decides from the URL which `AdminPage` should be loaded. The `AdminPage` creates a new instance of `ModulesHTML`, which accesses the XML file, and gets the module ids that belong to that page. It passes the module ids through to `DefinedModules`, which eventually returns an actual `Module` object (either a `ChartModule` or `TextModule`, depending on the given `type`). For this, `DefinedModules` creates a new `DefinedModule`, which contains the information to create and return a new Module. When `ModulesHTML` has obtained all modules, it checks for each module whether the user has the sufficient amount of rights to view it, it calls the `toHTML` method for these modules, and combines the returned strings in one big string (with HTML code). This is passed through to the `AdminPage`, to the `AdminController` and finally, the entire page is presented in the browser of the user.

Charts

When placing a module on a page, if a `chartType` was set in the module, a call to the script `chart.js` is made. This call contains the the id of the module, and the script obtains the data from the server belonging to this module. The script then plots the chart as required inside the module. The type of chart depends on the `chartType` of the module, and the dimensions correspond to those of the module. Finally, a interval can be set after which the charts are updated with newer data.

```php
<?php
/** Classname (and filename) is automatically the module's id.
 * Class should extend DefinedModule or StatDefinedModule */
class newmodule extends StatDefinedModule
{
    /* Visible title in the handle of the module on the page */
    public $title     = 'New module';
    /* Whether the module is wide (double size on the page) or not */
    public $wide      = true;
    /* Optional. If the module contains a chart, it should be set which type.
     * If not set: The module does not contain a chart. */
    public $chartType = DefinedModules::LINECHART;
    /* Defines which groups have access to this module */
    public $right     = array(Right::SOMERIGHTS);
    /* Optional. Whether the stats of this module in the database should be
     * refreshed when running the refresh-script. Not set = true */
    public $refresh   = false;

    /** Create interesting content for the module, set it, and return
     * $this->module in the end. The HTML tags should also be created here */
    public function getModule()
    {
        $content = createSomeContent();
        $this->module->setContent( $content );
        return $this->module;
    }

    /** Apply a DB mapper or more to obtain useful statistics
     * and store the results in $this->statsToSave. This method
     * is run when the stats for this module are saved in the DB */
    public function setStatsToSave()
    {
        $objects           = getObjects( $objectMapper );
        $this->statsToSave = calculateUsefulStatistics( $objects );
    }

    /** The DB stats of this module will be stored in $this->loadedStats
     * before this method will be run. Do some useful processing with the
     * data, depending on the given parameters, and return the result.
     * This method is used for creating charts in JavaScript (via the
     * controller), or in getModule() for textually presenting the data */
    public function getProcessedStats( $parameters )
    {
        $result = performSomeActionOn( $this->loadedStats, $parameters );
        return $result;
    }
}
?>
```

Figure 13: Code example of a DefinedModule subclass

```
<pages>
    <page>
        <name>some_page</name>
        <modules>
            <module>existingmodule</module>
            <module>newmodule</module>
        </modules>
    </page>
</pages>
```

Figure 14: Example of the XML file's content, with a page and two modules

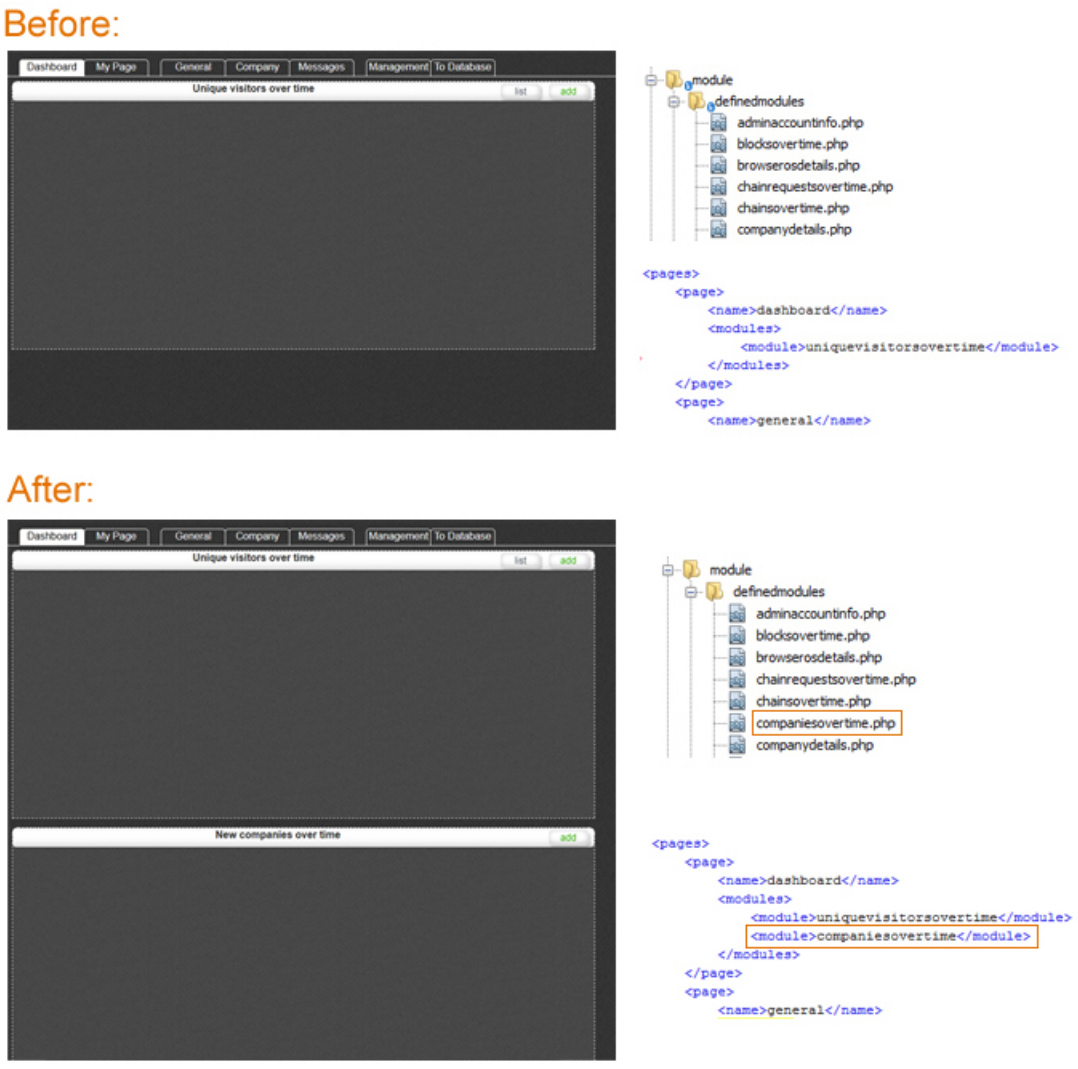

Figure 15: Visual representation of adding a module

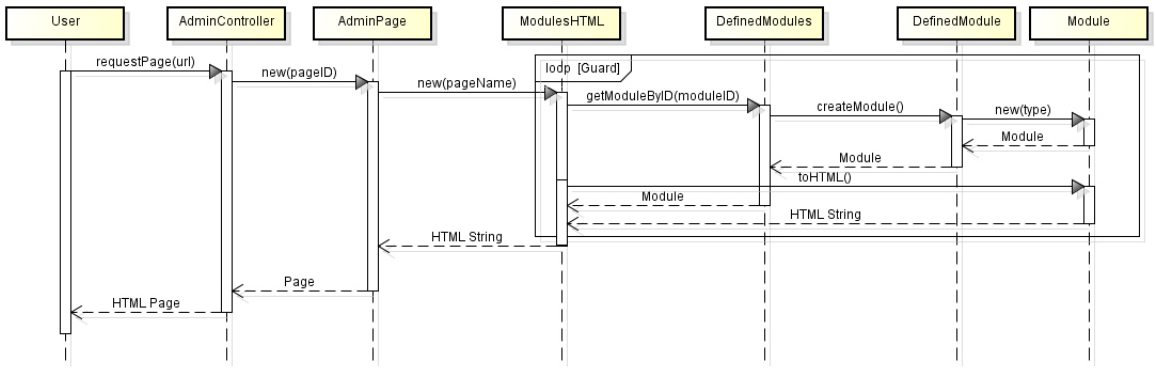

Figure 16: Sequence diagram of a page load with modules

5.3.2 User Management

As said, each user of the OAT has its own account. These accounts contain the data regarding the My Page and the rights of that account. When trying to see a module and when entering the OAT the right of the logged in user are checked. Having a right is defined as belonging to a certain group, these groups provide access to certain elements in the OAT. If the user and the module he wants to see both belong to a same group, the module is shown. If not, the module is withheld entirely from the user. Some tabs in the navigation also require a certain right too see them, if the user does not have this right, the tab is also hidden.

When a new CHAIN*els* employee uses his CHAIN*els* account to log in to the OAT, a new AdminAccount is created. This AdminAccount is granted access, but with minimal rights. This means no modules are visible yet, and the Management page and access to the database are not available. When a System Admin, the person with full management rights, gives the new AdminAccount more rights, more modules become visible.

5.3.3 My Page

The My Page is different from all other pages, as the XML file with all defined pages is ignored, and the list of module ids is obtained from the user's AdminAccount. The modules are then processed as normal.

5.4 Front-end

Figure 17 shows a screenshot of a My Page in the OAT. This design has been created by CHAIN*els* designer Willem Buijs.

In this screenshot, you see:

1. The live feed. This is always visible in the top right corner of the screen. It displays data about today.

2. The navigation bar. Here menu buttons are grouped together by: special pages, normal pages, and management pages. The My Page is currently selected, as the background colour of the tab is white.

3. A normal module. Is has a (white) handle bar, which contains the title and a button. The content of this ChartModule is a chart and the filtering options for the data. The bottom module on this page shows a double width module. The width of the chart is automatically changed to make use of the extra space.

Figure 17: Screenshot of My Page in the OAT

4. As the rightmost small module is dragged to change the layout, a placeholder is shown to indicate where the module would be placed. Dragging a module is only possible by its handle to prevent clicking events from becoming dragging events.

5. When viewing any other page than My Page, this buttons allows the user to add the module to My Page. When on My Page, this button deletes the module from My Page.

Interface
The designer opted for a design with minimal use of colour. Colours are still used in the button in a module's handle, and to easily distinguish links, which are also underlined. This provides a high contrast design while eliminating all problems which may occur with the use of lots of colours. This also makes for a design which is easy on the eyes. By using modules on all pages, with the same design, consistency is maintained.

Modules
All information in the OAT is shown in modules. This brings a size restriction, but this restriction forces the information to be compact and therefore easy to comprehend. No large complex contructions exist in a module. All management options are also placed in a module, not only to enable modularity, but also to increase consistency throughout the OAT.

The layout of the modules can be changed on any page, by dragging the modules around. This is done with the following lines of code:

```
window.onload = function(){
    $( "#modules" ).sortable( { handle: '.handle', cursor: 'move',
        placeholder: 'module-placeholder'} );
```

The function `sortable()` from the JavaScript library jQuery makes any the inner of the object on which it is called sortable. This means the children of an element can be sorted by a given function, thus when called on the body of the page, `#modules`, the modules become sortable. With the added mark-up style:

```
.module{
    float: left;
}
```

the modules are forced to the top-left of the page. The modules line up next to each other if there is room left on the top row, and create a new row if there is not. Many website, including CHAIN*els*' site, have a fixed-width viewport to maintain compatibility with lower screen resolutions, while having a lot of unused space on higher resolution screens. While a minimum of two small modules next to each other (or one double-width module) is the minimum width of any page, enlarging the browser screen to full-screen on a full-HD monitor enables 4 small modules to align next to each other. While this minimises the waste on larger screens, it also enables a user to drag modules on the same page next to each other to compare interesting data. Although this layout is default when visiting a normal page, on the user's My Page he can choose his own layout and save it to his account.

Charts
A chart displays the data from the CHAIN*els* system. This information is usually from a set time period, which the user can change. As for most objects the companies id is set in the database, the user can filter the objects on a company level. There are two types of charts available: line charts and bar charts. The charts in the OAT were drawn using the library *d3.js*. After a module containing a chart has been placed on a page, a call is made to the `chart.js` script to create the chart. This call contains the module id, whether the module is single or double width, and the type of chart. With the module id the script obtains the chart's dataset via a call to the server. Any parameters, such as the period to search in, are passed along with this call. From the dataset

the axes are determined, and the lines or bars are plotted. Finally, the charts is placed in the module so that it is visible to the user.

The charts update at a set interval to display fresh data if the user did not change the page. Alternatively, if the user wants to filter the data, he can update the shown chart manually. Although the defined module has a preset chart type, the user can switch back and forth between a line chart and a bar chart.

6 Testing

6.1 Performance test

To test whether the OAT was future-proof, large test sets were created to assess its loading times. The loading time is defined as loading a page in the OAT with the module which required heavy calculations. For this test no other modules would be on the page. These test sets would consist of a large number of companies.

As for almost all charts the company ids from the objects are stored, retrieving the company objects when viewing a module would be the most intensive task in the OAT. Therefore we decided to test this aspect of the OAT. When two objects belong to the same company, the company id will be stored twice in the database. However, as CHAIN*els* uses a caching technique to prevent the retrieval of the same object from the database multiple times, it is the maximal loading time of all companies which determines the speed.

Creating the test sets

Two test sets were created. The first would consist of 10.000 companies, the second of 20.000. Each company would be complete, i.e. all the information would be set. For all companies, all the information was the same, except for their industry and status. The industry of a company was randomly chosen from all available industries. This gives an almost uniform distribution across all industries, which is not realistic, but sufficient for our need. The status of a company indicates how much information they have stored in the database, and although this was the same for all companies, all statuses were overwritten by either a 3 or a 4, randomly chosen. Again, this is not realistic, but sufficient for our testing purposes.

There are two situations where companies are retrieved from the database. The first is when the data in a chart is filtered on a company level, the second is when a management module need the list of all companies. In the first situation, it is highly unlikely all companies are retrieved. For example, when viewing a chart about messages or chains, only the companies which actually posted a message or made a chain in the given time period are retrieved. In the latter situation, all companies are always retrieved. For both datasets, loading any chart for the first time is almost instantly, as no objects are required from the database. This ensures responsiveness when navigating through the OAT.

All tests were performed on a modern desktop computer. The relevant specifications of this computer are: Intel i5-2500 processor, 8GB RAM memory, 128GB SSD. On the computer Windows 8 was installed, the tests were performed with the Mozilla Firefox browser, and the server was installed on the SSD. All times were measured via the console in the browser.

Results

The test results are shown in table 1. It shows the time it took for the page to load in seconds. We made sure only one module would decide the loading time by placing this module on the My Page.

Looking at the results, a database with 10.000 companies may very well be workable, although it may be a viable option to start storing more information in the database. At 20.000 companies, loading a page becomes slow. This is a point where the current way of calculating data is no longer sufficient.

Action	10.000 companies	20.000 companies
Filtering on industry	6.1	12.9
Retrieving companies list	7.1	14.2

Table 1: Times in seconds for tested scenarios

6.2 Acceptance test

For the last ten days of the project, the OAT was deployed in the CHAIN*els* system. This gave the CHAIN*els* team some time to get to know the system, to check for possible errors, and to see if the OAT satisfied their needs. In this period, all errors were reported automatically by email to our supervisor, so we could fix them. Some errors came up during this period, which we quickly fixed. It also became clear the OAT lacked some functionality. This was added in the last phase of the project, while creating this report.

7 Requirements evaluation

Most of the set up requirements in Section 3 have been fulfilled in the OAT. Certain requirements have not been fulfilled. Some of those with the lowest priority are not in the OAT due to a lack of time, but some are not implemented due to a change of design. These decisions and design changes are discussed in this section. An evaluation is given of the difference between the initial design and the actual implementation. Therefore, the requirements that are exactly implemented as designed, and do not contain nontrivial implementation, are not mentioned.

- *The number of gained credits (total or per time period) should be dividable per action with which they were gained, for both real time and total numbers throughout the OAT.* and in Management: *The amount of credits which are rewarded for certain actions should be changeable.* It is possible to see the gained credits in a chart. However, as the credits are not fully used yet in the system, a credit transaction does not yet contain information about the reason why it was created. Therefore, we could not implement this, but it was not a high priority.
- *For both chains and invites, the average time between requests and accepts should be available.* and *For both chain requests and invites, a percentage of accepted requests should be calculated.* We did not measure this time. A problem was that chain requests disappear as soon as they are accepted as chains. Therefore, the time that the chain was requested cannot be found anymore in the system.
- *The number of messages should be dividable per type of user throughout the system.* and *The pages a company views and the messages a company posts should be dividable in types of user accounts.* Currently, every company in the system has only one user. Therefore, implementing this was not relevant yet. item *The average amount of recommends and comments on posts. The amounts should be dividable in type of message and in company number of employees, region, and industry throughout the OAT.* As the amount of comments on posts is currently not very high, these numbers would be near zero. Therefore, it was not really relevant to implement this.

Per company:

- *The amount of time it took the company to complete the sign-up process.* and *The time they spent in the introduction manual of the website.* These times are not saved in the system, but was eventually not seen as relevant data.
- *A rating to show how active the company is.* In our tool, this is only measured by the number of chain requests a company does. This could be extended in the future.

Per page:

- *A visitors flow, showing where the user came from and what the next page the visitor goes to is.* This would have costed a lot of time to implement, which we eventually, unfortunately did not have.

Per user:

- *The amount of pages the user visits per session and on average.* We only provide information about all page visits together. Unfortunately, a separation of distinct pages is not provided, due to a lack of time.

Visualisation requirements:

- *Visualisations of the data should be zoomable where possible, to increase detail of visualisations on the fly.* Our charts are not zoomable, due to a lack of time.

Management requirements:

- *Companies' requests to be featured on the homepage should be handled through the OAT.* This is not registered in the system. Instead, an email is sent to CHAIN*els* when someone does a request. This does not enable us to implement this.
- *Feedback from the companies should be viewable.* Not implemented, not in the system yet and lack of time.
- *Error reports from the site should be viewable.* and *Notifications should always be visible to inform users about important events, like an error report.* Not implemented, lack of time.

The nonfunctional requirements are harder to evaluate. We think that we did a good job in succeeding these requirements:

- *Fit in existing framework.* We used the same package structure as CHAIN*els* , and used their database mappers and other objects. We think it is easy for the CHAIN*els* team to understand our code, when they know their own code.
- *Modularity.* As can be read in the implementation, it is very easy to add a new module. This was a requirement with high priority, and we think we succeeded well.
- *Speed and ease of use.* We managed to find a suitable balance between processing data before and after storage, which provided us enough data analysis opportunities, and was still fast enough to use. Also, the modules and charts are clear and do not require a manual.

8 Recommendations

In the end of the project, it is quite clear how the system is built, and which things are not present in the system yet. From this, we can give recommendations for the future. In this section, future recommendations for extending and adjusting the OAT are discussed, as well as recommendations for the company.

8.1 For the OAT

- For almost all charts, an array of company ids per date are stored. This is how we wanted it, but has it limits. It would be useful to store account ids instead for some charts. When a company can have more than 1 account in the future, it will become interesting to obtain information from a specified user account.
- When the statistics in the database is refreshed, everything is refreshed. However, data from the days before is unlikely to change often. It would be much more efficient to add the data from the current day to the old data, and update the older data less often.
- The OAT contains two different types of charts, a line chart and a bar chart. More chart types would be useful, for example a pie chart, for the different browsers that visitors use.
- A more sophisticated method to measure the activity of a company would give interesting insights. Instead of only counting the chain requests, also the frequency in logging in and viewing pages could be taken into account.
- The processing of statistics is mainly based on separate functions, but a sophisticated class structure that deals with all statistics together would be much more powerful and extendable, as it can deal with different types of statistics when built correctly.

8.2 For CHAIN*els*

- It would be useful to create a way in which companies can be obtained from the database, but in a smaller format, containing only the most important information. This leads to a major decrease in time when a large number of companies is requested. The same could be applied to pageviews.
- A request from a company to feature at the homepage could be stored in the database, instead of only emailed to CHAIN*els* . This way, the OAT could process these requests. The same applies for error reports and feedback from companies.
- For a new chain, it would be useful to also store the time when the chain was requested. In this way, the length of time could be measured between request and acceptance. A total average and other interesting statistics could be obtained from this.

9 Conclusion

We can look back at a successful project. We have created a tool that enables CHAIN*els* to get insights about the usage of their business social platform. Also, we created new management functionality for the company. We combined this in a safe and easy-to-use web-based tool.

The project allowed us to gather useful experience in requirement analysis and specification, planning, implementation, and relationships with customers (in our case, CHAIN*els*). Among the learnt lessons, we understood the importance of a detailed and agreed upon requirement specifications phase. It also allowed us to improve our abilities in recognising priorities in a specific project.

For a next project, we would spend more time in designing the system beforehand, including its classes and relations between them. This could have saved us time, as we rewrote our code on a number of occasions. Also, we would have created a more intelligent statistics class, being able to process different kinds of statistics data, which would save us efforts in using functions for each different type of data.

We experienced a very satisfying collaboration with the company. They provided us with all the help that we needed for creating the product and the project documents. We have learnt a lot from their skills. We think that we became more experienced in engineering software, and will use this experience to perform even more successful projects in the future.

Appendices

Appendix A: Problem Description

Project Proposal
BSc Technische Informatica

offered by

CHAINels

Burgwal 47
2611GG Delft

www.chainels.com
0153642667
post@chainels.com

Technische Informatica

Bachelor Project

Faculty of Electrical Engineering, Mathematics and Computer Science, Mekelweg 4, 2628 CD Delft, The Netherlands

BachelorprojectTI-EWI@tudelft.nl
TI3800 (15 ECTS)

Contact and Coordination

Martha A. Larson
M.A.Larson@tudelft.nl
http://homepage.tudelft.nl/q22t4/

Hans-Gerhard Gross
H.G.Gross@tudelft.nl
http://swerl.tudelft.nl/bin/view/GerdGross/WebHome

Company description

CHAINels – your business discovered

CHAINels is een digitale platform dat hulp biedt, specifiek aan het MKB, op het gebied van online business op lokaal niveau. Dit houdt in dat CHAINels haar site als middel inzet om lokaal zakendoen te stimuleren.
Op het platform kunnen bedrijven kosteloos een profiel aanmaken waarna ze onderling kunnen 'chainen'. Dit 'chainen' betekent dat bedrijven onderling een connectie aan maken om zo een netwerk te vormen met hun zakelijke relaties. Ook biedt CHAINels bedrijven business intelligence en ondersteuning bij het centraal managen van hun social media. Dit gebeurt door middel van overzichtelijke en eenvoudige automatisering (visualisering van statistieken). Bij dit onderdeel kan men waardevolle informatie over hun huidige en potentiële CHAINs bekijken. Het centraal managen van de social media kan gedaan worden vanaf de zogenoemde 'Desk' op het platform. Hier kan men berichten op een eenvoudige manier vormgeven en plaatsen op hun social media-account.

CHAINels heeft in 2011 als open-bèta zijn oorsprong. Vanuit de TU Delft is het concept als studieproject ontwikkeld tot echt product. Hierbij waren het studenten Erwin Buckers en Vincent Koeman en docent dr. Shahid Suddle die de potentie van CHAINels inzagen. Zij hebben het product geperfectioneerd en uiteindelijk vond er op 11-11-11 de lancering plaats.
Na enige tijd bleek de open-bèta versie toch niet te voldoen aan de wensen uit het bedrijfsleven en van het team zelf. Besloten werd om in 2012 een doorstart te maken, maar het product geheel te vernieuwen. Hierbij is het team terug naar de basis van het product gegaan en is alles opnieuw opgebouwd. In deze periode is er ook flink wat versterking bijgekomen. Zo kwam werd er een Product Developer toegevoegd, een Business Developer en een extra ICT'er. Met dit team kwam CHAINels, de bèta-versie, van de grond en begonnen bedrijven zich in te schrijven.
CHAINels ging het zelfs zo goed doen dat er anno 2013 sprake is van een 'Angel investment'. Waardoor CHAINels haar team heeft kunnen uitbreiden en momenteel gevestigd zit in het centrum van Delft.

Project description

De opdracht die wij aanbieden is het ontwerpen en implementeren van een online administratietool voor de werknemers van CHAINels. Via deze tool moeten accounts van bedrijven worden beheerd, gebruikersstatistieken kunnen worden bekeken en geanalyseerd, verschillende soorten berichten worden verstuurd, en meer.
Sommige van deze losse functionaliteiten zitten al in ons systeem ingebouwd; de nadruk zal daarom liggen op het ontwikkelen van een platform dat deze functies integreert, vormgeeft en dat uit te breiden is in de toekomst. Ook zal er rekening moeten worden gehouden met de verschillende rollen van werknemers en bijbehorende bevoegdheden.

Auxiliary Information

CHAINels is geprogrammeerd in PHP (object georiënteerd), Javascript (jQuery/AJAX) en HTML, dus ervaring met deze talen is vereist. CHAINels draait op een Redis database (NoSQL), dus ervaring daar mee is een voordeel, maar is niet vereist.

Verder zal er veel begeleiding en hulp zijn vanuit het bedrijf, tevens worden er werkplekken aangeboden en is er al een gangbare structuur ontwikkeld om op professionele wijze het project te voltooien.

Vorig jaar zijn al twee teams voorgegaan die het project met succes hebben afgerond.

Project team

Supervision

Name Company Supervisor	Email	Phone
Vincent Koeman	Koeman@chainels.com	
Name TUDelft Supervisor	Email	Phone
Alessandro Bozzon	a.bozzon@tudelft.nl	

Student team

Name	Study ID	Email	Phone	Prerequisites* fulfilled signature
Henk Kant	4024400	henkkant@hotmail.com	0640282668	
Daniël Mast	4092457	danielmast92@gmail.com	0628658613	

*Prerequisites: by signing this box, as a student, I declare that I fulfill the prerequisites of the TI3800 bachelor project as stated in the digital study guide.

Appendix B: Planning

Our project is split up into four phases: an orientation phase, a design phase, an implementation phase, and a reporting phase. These phases last 2, 2, 4, and 2 weeks, respectively. For most of the phases there is an overall planning. However, as we were just a group of two people, we do not plan the finer details. Working closely together, we are able to discuss the details when necessary. Because of this, there was not always a clear division of tasks. Some larger tasks will be picked up by one of us, others are done together.

Orientation Phase
In the orientation phase, the task was to first create a plan of approach. This is a relatively small report and no clear task division is made. The first two days of the orientation phase are spent on the plan of approach, the remainder is spent on the Orientation Report.

For the Orientation Report (OR) we spend time looking at what we needed to know before we could start with the tool itself. We research two subjects: Web usage mining (WUM) and how to efficiently create a user interface. The report is split into six subtasks:

1. survey of existing analysis technology
2. survey of existing libraries
3. web usage mining research
4. visual representation research
5. survey of the CHAIN*els* system
6. survey of the collected data

Item 1 and 3 are on the subject of web usage mining, and assigned to Daniel. Item 2 and 4 are related to creating the user interface and are assigned to Henk. For items 5 and 6 we need to study the existing CHAIN*els* system and get to know the database. As it is important for both of us to be comfortable working in this environment, we both spend time on those two tasks. The research consists of finding relevant material and creating a report out of it, which we spend 8 days on. This leaves two days to get to know the CHAIN*els* system and database.

Design Phase
In the design phase we need to create a requirements document and the general design of the new system. This phase will take two weeks.

The first week we created the requirements document. This involvs working together with the CHAIN*els* team, after which we can write up the document. There is no clear division of tasks on this report. This report requires a couple of versions, which are spread over the week.

In the second week we create a design for the system. The interface is mostly set up by Henk, with ideas from Daniel. The draggable features and the charts are mostly designed by Henk. For the back-end, Daniel creates a design for the storage in the database and the modules. The remaining smaller components, such as user management and the required web pages, are created by Henk.

Implementation Phase
For the implementation phase we require a better planning. From the tasks we identified we created a Gantt chart. Figure 20 shows this Gantt chart.

In the implementation phase, Daniel is most responsible for the retrieval and storage of the information. He also creates the live feed and the first setup for the modules. Creating the interface is Henk's task. This included creating the pages, the necessary controller, and the charts. Logging in to the OAT is created by Daniel, and Henk will refine this by storing AdminAccounts in the database, as he has knowledge of this process because of the storing of pages. Interaction

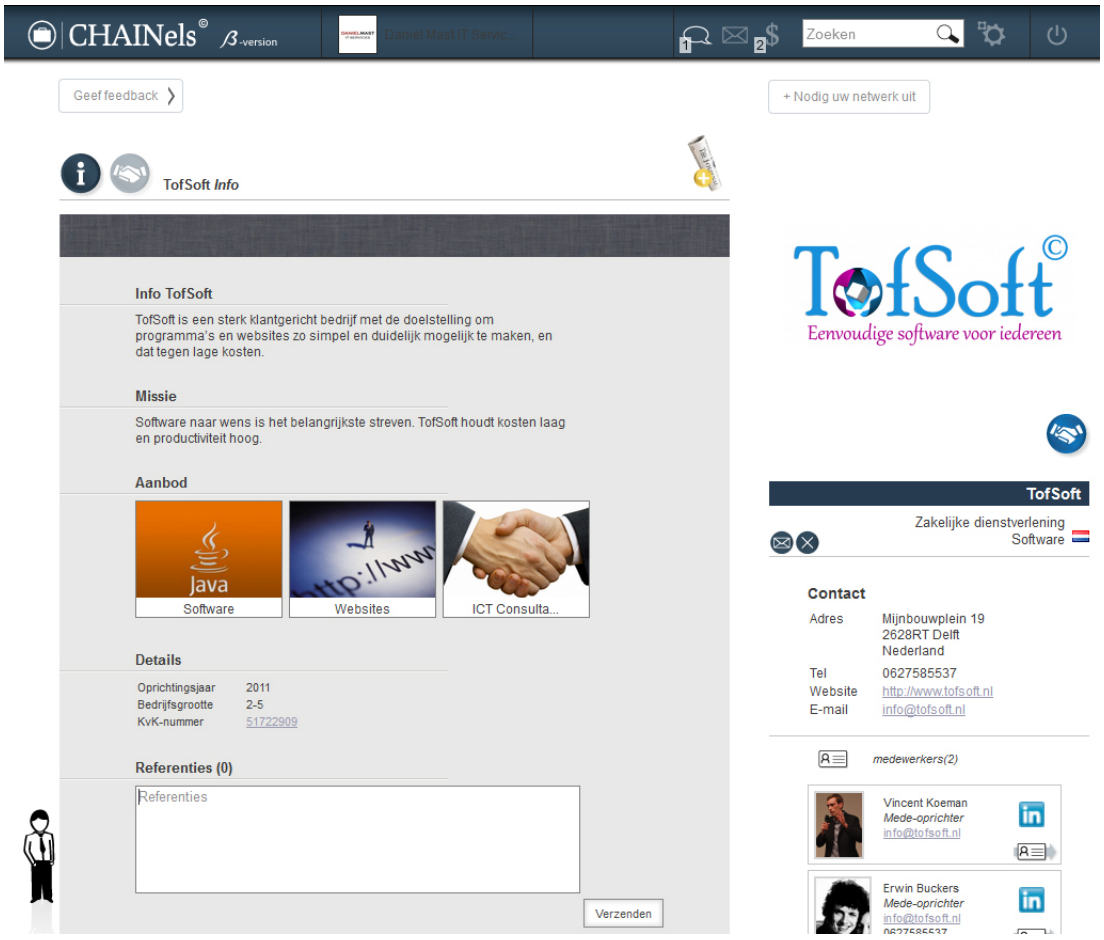

Figure 18: About-page of company TofSoft

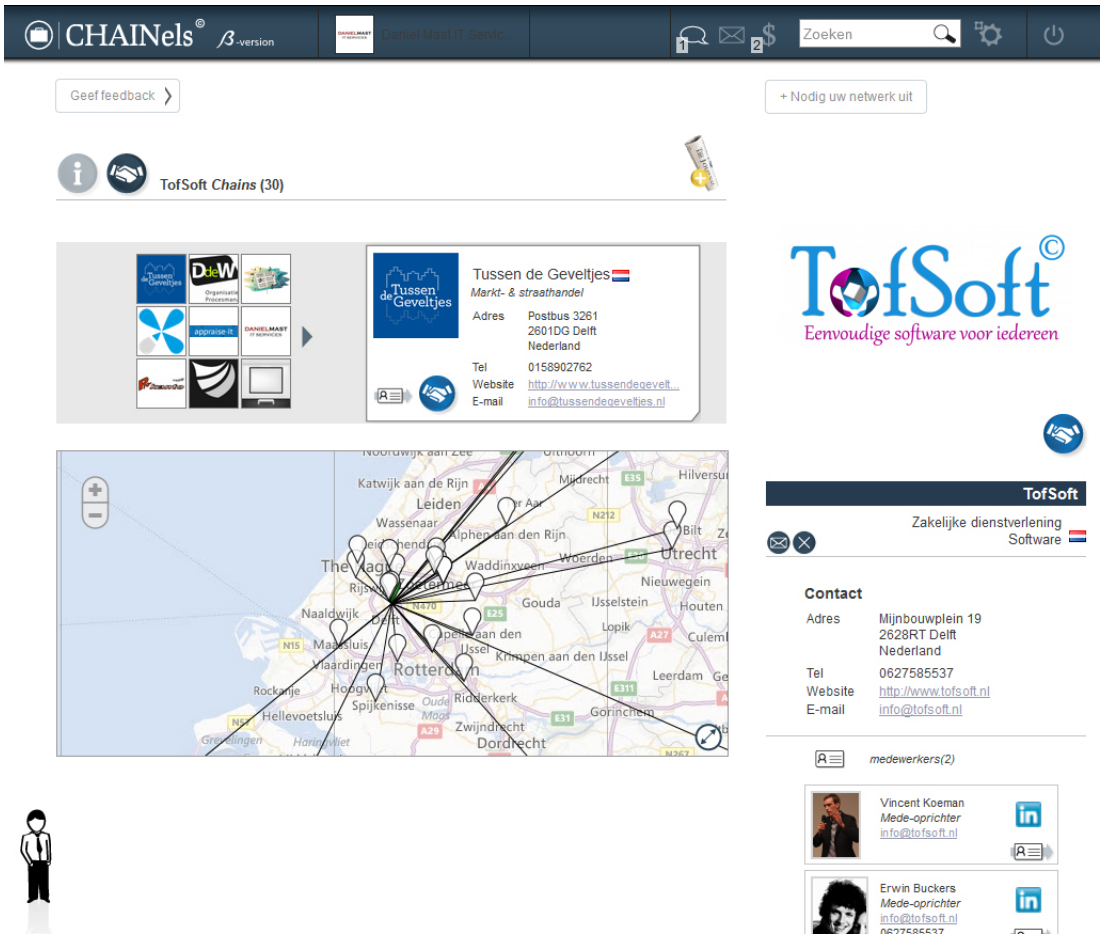

Figure 19: Chains-page of company TofSoft

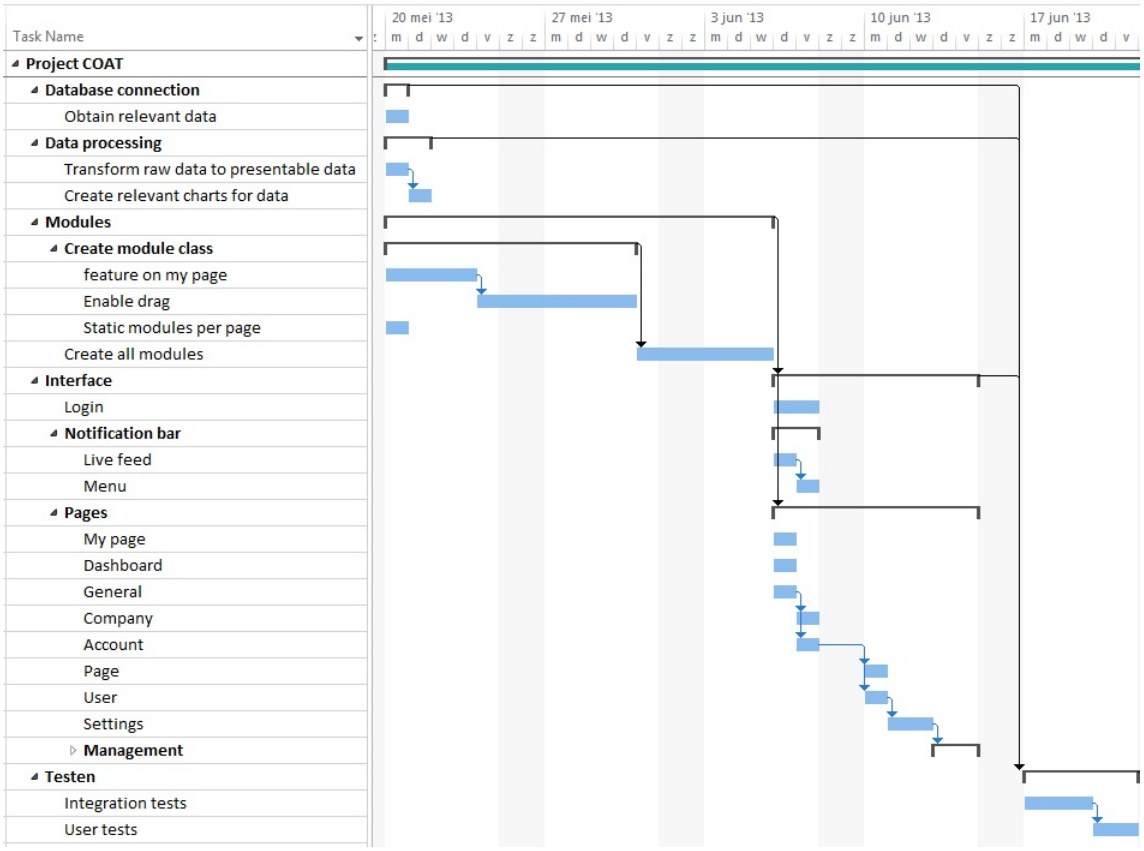

Figure 20: Gantt chart with the planning of the implementation and reporting phase

in the front-end is handled by Henk.

Changes in the planning

Although the basis for the objects were laid out as described, from a certain point on we both worked on the system as a whole. In the end, while Daniel laid out most of the database access, some improvements were implemented by Henk. In the last days of this phase, any missing functionality was filled in by Henk, while Daniel took it upon him to improve the way module were created in the system, and changed it to the loading from the files which is the final implementation.

We laid out an initial planning, but it became clear details lacked, and the order of the tasks and the time we spent on them changed.

- In the first week of the implementation phase storing the information and creating a first interface were done. In the second week the charts were created from the gathered information. At this time the way the information was stored was improved and the live feed was created. That week the AdminAccounts were also created.
- Although quite early a basis for the modules was defined, working modules on the pages were introduced in the latter half of week two. In week three, charts were redefined. Some time was spent on using object oriented JavaScript and the updating of the charts.
- In week three we made great improvements in the infromation storage causing improvements in processing times. This week we also started defining more modules and the information they needed.

- Week four was used to add functionality to the charts. This also caused renewing the way the data was stored, going from just number to objects. At the end of this week it became clear the way modules were defined and placed on the pages needed to be redone. This caused running into the first week of the next phase.

Reporting Phase
With the implementation phase taking longer than planned, there was less time for the reporting phase. As the report consisted of a number of distinct topics, tasks were devided. In the first revision the Problem analysis was done by Henk, and in a later revision reworked by Daniel. The requirements and requirements evalutaion was done by Daniel. The description of the design and the latter half of the implementation was described by Henk, while Daniel described the implementation of the stored data. The testing of the OAT was performed and documented by Henk.

Appendix C: Plan of approach

Project COAT - Plan of Approach

Daniël Mast (4092457)
Henk Kant (4024400)

April 23, 2013

Contents

Foreword

This document contains the plan of approach of our bachelor's project at the company of CHAINels. We will discuss what role we are planning to fulfil for the company and which boundaries we have to take into account. Moreover, we will describe what is expected from us as project members, as well as from our company's supervisor, and our coach from TU Delft. Also, we will discuss how we will split the project into phases, and how we will manage deadlines and appointments.

Summary

In this project an online administration tool will be created for the website of the company CHAINels. CHAINels' website supports companies in their online presence. This project will be active for the next 10 weeks. This document is the plan of approach for this period. The project team will orientate extensively to make the product as good as possible. The team will work closely together with the employees of CHAINels to have a clear view on the goals of the project and the requirements of the product.

1 Introduction

In the end phase of the bachelor Computer Science, students execute a project at a company. We have chosen for the company CHAINels, and already had contact with employees of this company. This company also has an agreement with TU Delft, by providing students a possibility to do a bachelor's project. We chose this company, because we have great interest in what this company does, and would want to add features in their system.

We are Daniël Mast and Henk Kant, two Computer Science students in the end of their bachelor phase. This plan of approach is a first step in clarifying what we are going to do in the given period, and how we are going to do it.

1.1 The company

CHAINels is a company located in Delft, currently aiming at small and medium enterprises (SME). CHAINels supports companies in their online presence. It enables companies to create a profile on www.chainels.com, where they can establish new contacts with other companies. CHAINels can also provide tools to these companies, for example the possibility to search for companies of a certain field in their area.

1.2 Structure of the document

In Chapter 2, the properties of the project will be described. The environment of the project, its goal, deliverables and requirements will be discussed. In Chapter 3, a planning of the project period is given, together with the collaboration method that is used. In Chapter 4 will be discussed how the project is designed. This includes the people that work on it, what they do, which tools they use and how they report their progress. In Chapter 5, the measures that are taken in order to assure the quality of the project are given.

2 Project Description

This chapter describes what is expected of the project. It will mention the stakeholders, the problem, the goals and deliverable, and finally the conditions to which the project must comply.

2.1 Project environment

The CHAINels network has an ever increasing number of companies using the network. To get an idea of how these companies use the network, data on almost every click on the website is recorded. This information consist of many aspects which can provide useful information if it is presented in a useful manner to CHAINels marketing team. Unfortunately, this information is just stored as raw data and there is no use for it as is.

2.2 The goal

The goal of this project is to create an online administration tool (OAT), where the people of the CHAINels team can monitor the accounts of their users and the network as a whole. This monitoring consists of the growth over time and how new users learn of the network, the activity of the users divided into overall activity and specific aspects of the network, and the connections made amongst users. The information the team can extract from this information gives pointers to which functions of the network are rarely used, how much time the users spent and what they do, and through which channels most companies get to know the network. This gives opportunities to improve functions or increase marketing efforts where needed.

2.3 Problem description

The project team is expected to design and implement an OAT which the CHAINels team can use to extract knowledge from the stored information. The information should be displayed in an efficient manner, such that employees of CHAINels can effectively act upon this information. This information has to be highlevel to be hypothetically displayed on a monitor in CHAINels' office, as well as extremely detailed to make proper analysis possible. It is not expected of the project team to extract data from user activity. If additional data is required the CHAINels team provides this information.

2.4 Deliverables

In the end, the team delivers the OAT along with the program code. Additionally, sufficient documentation should be provided to give the CHAINels team the possibility to expand the tool in the future. Documents regarding the plan of approach and the first phase of orientation will be created in the first two weeks of the project. To make implementation possible a list of requirements will be created after consultation with the CHAINels team.

2.5 Requirements

The CHAINels team already defined some requirements for the tool:

- Security: Only the CHAINels team should be able to use the OAT, security should be provided through authentication.
- Scalability: Currently the CHAINels team consists of 10 people, but the OAT should be able to handle larger groups of people people using the tool at the same time. This ensures the tool will be usable if the company grows.
- Modularity: After the tool is deployed additional features may be added in the future. The tool has to implemented in a way to make these extensions and additions possible.

- Stability: The OAT will be used every day and has to be tested extensively to ensure stability.
- Cross-browser compatibility: It is important the OAT will work in the latest version of any modern browser (i.e. Chrome, Internet Explorer, Safari, Firefox).
- Usability: The tool is possibly used by anyone in the CHAINels team; the tool should require no technical skill to be used efficiently.

Additionally, the project team identified other requirements:

- Efficiency: The OAT should be fast and using resources efficiently.
- Robustness: When temporarily operating outside of normal operating parameters, the tool should degrade gracefully and not fail instantly.

2.6 Conditions

To make the OAT successful some conditions need to be met.

- The project team members need adequate skills with the programming language used to create the tool. This ensures the tool will be efficiently implemented.
- The project team members should use knowledge obtained during other bachelor courses in order to write clean code and thoroughly test the tool before deployment.
- The requirements should be carefully defined to ensure dependability and usability.

3 Approach and Planning

This project spans the minimum of ten weeks, for which the following planning is made:

- In the first half of the first week a plan of approach is created.
- In the first and second week research will be conducted and an orientation report will be created. During this phase appropriate techniques will be researched, along with alternatives for used tools.
- In weeks three and four specifications will be gathered and the design of the OAT will be thought out. In this phase the specifications should be completed and ready to continue with implementation.

- Starting in week four up to week 9 the implementation will be done. This means writing actual code to create functionality in the OAT. The CHAINels team expressed a desire to use an early version of the OAT as soon as it is available. Testing is done throughout this phase by use of unit tests.
- In week nine and ten extensive testing is done. Besides unit testing throughout the implementation phase integration and system tests will be conducted. User tests will be done with the CHAINels team. Any improvements to be made to increase usability and stability will be conducted in this phase. All remaining documents to conclude this project will be completed in this phase.
- The project ends with a presentation by the project team for the project coordinator, the project coach, and the company supervisor.

For this project an agile approach will be used. This means work will be conducted in sprints, with every sprint adding another piece of functionality. Functionality will be prioritised, with functionality which adds most value implemented first. A backlog will be created which contains all items to be implemented. Additionally, on a more frequent basis very short meeting will be held to keep everyone in the team up to pace.
Every week a meeting with the company supervisor will be conducted to ensure the progress of the project team and the quality of the OAT. Meetings with the project coach will be conducted on a less frequent basis. Before June 14th the program code will be sent to the Software improvement group (SIG) for inspection and recommendations for improvements.

4 Project Design

4.1 Organisation

The stakeholders in this project are:

- Project team: Henk Kant and Daniël Mast.
- Company supervisor at CHAINels: Chief Technology Officer (CTO) Vincent Koeman.
- Project coach: Assistant Professor Alessandro Bozzon, currently active in the Web Information Systems department at TU Delft.
- Project coordinators: Martha Larson and Gerd Gross.

4.2 Personnel

In 10 weeks, the project team will work on the project, working closely together with Vincent and the other employees of CHAINels, and the project coach. By

discussing with each other, this group will be able to state the requirements of the OAT, and produce it.

4.3 Administrative procedures

CHAINels already uses an extensive packet of software and web tools to guide their work. The project team will adopt most of these.

- JIRA: A web tool for project management, which also supports Scrum. This will be the main tool for managing all tasks.
- SmartSVN: A client for SVN. CHAINels stores a great amount of the website and other documents on an SVN repository.
- Google Drive: The project team mainly stores important documents here that are only relevant for them and not for CHAINels. Therefore, these documents are not stored on CHAINels' SVN.
- Jenkins: A continuous integration tool. This tool is used for continuously and automatically testing the code that is being worked on in this project.

4.4 Reporting

The project team will spend most of the time during the week at CHAINels' office. This enables continuous consultation. Every Tuesday, the project team and the company supervisor arrange a meeting to discuss the progress.

4.5 Resources

The project members use their own laptop for development and to fulfil other project tasks, like writing the reports. The IDE in which the project team will develop the OAT is NetBeans. A useful browser plugin that will be used is Bonfire. This tool enables appointing bugs on the website by making a snapshot and writing a comment, and adding this to the tasks in JIRA.

4.6 Additional tools

CHAINels uses object oriented PHP to build the entire website, alongside programming languages like HTML, JavaScript, and CSS. To maximise campatibility the project will also be carried out using these programming languages. The allready-in-place Redis database will be used to access the stored data. To create the neccesary visualisations an undefined existing library will be used.

5 Quality Assurance

To ensure the quality of the OAT and the procedure of the project, different measures are taken. First, it is very important that the product is tested frequently. Jenkins will do automated (unit) tests, but in addition it will be critical

to do user tests. Therefore, it is convenient to produce beta versions of the product, so that it can be tested before it is finished, and modifications can be done before the end to produce a better result. In addition, after eight weeks, the Software Improvement Group (SIG) will test the code and measure its quality, amount of comments et cetera. This brings us to the second measure: documentation and commenting of the code. In order to keep the code understandable and editable, it has to be documented well and commented thorougly. Not only the code should be documented, but also the progress of the project, in order to keep clear what we are doing and where we want to go. The orientation document that will be written in the first two weeks is an example of this. Last, the product data is protected from loss, because it is stored on the repository, which enables version control.

Appendix D: Orientation Report

Orientation Report

Henk Kant - *4024400*
Daniël Mast - *4092457*

May 7, 2013

Contents

Introduction

In this report we perform an orientation for our project, in which we will develop an Online Administration Tool (OAT) for the company CHAINels. This is to get clear view of what our possibilities are during this project. We look at the company CHAINels itself, and the tools that they use. We also try to define some core aspects of our project and research them.

In the first section we start by giving a short description of the company CHAINels and how it started. In the second section we look at the tools the company uses to create and manage their system. This section gives a short description with just some basics which may be useful to our cause. In the third section we look at the existing CHAINels website. We try to determine how it is set up, we look at the data that is already collected, and how it is stored. The fourth and fifth section are used to take a look at techniques and research conducted. First we look at web usage mining and the techniques that already exist. In the last section, we look at efficient ways to create user interfaces and existing solutions to visualise data.

1 CHAINels

The company CHAINels started out as a bachelor end project by Erwin Buckers (currently CEO) and Vincent Koeman (currently CTO), supervised by Shahid Suddle. The idea in short was to create a social network for companies. On this network, accounts would represent companies as a whole instead of individuals, like on LinkedIn. Companies with smaller budgets and less knowledge of how to present themselves online, like small and medium enterprises (SME), may have difficulty with the Internet, and CHAINels seeks to provide an easy platform for online promotion. Businesses can make connections, called *chains*, with other companies, representing a business connection they have. The network also has a strong integration with the social networks Facebook, LinkedIn, and Twitter to increase the ease of use for companies by sharing messages not only on CHAINels itself, but also on those other networks. After releasing the network in New York late 2011, it has been in continuous development. New functionality is continuously added to increase the effectiveness for the users.

The team grew from three to eight members, excluding interns and ourselves. The team spans areas including business, economics, design, and marketing. CHAINels also gives students from a number of schools opportunities to be a part of the team, to create new functionality as part of a project, or to gain experience in the field.

CHAINels currently saves some information about the actions of the users of the website. This information is required by the marketing team go get some insight into the effectiveness of their marketing efforts, and by the development team to see which functions are used less frequently and may need improvements. Our assignment is to create an Online Administration Tool (OAT) that takes the raw data that is collected and represents it in an useful manner.

2 Tools to use

The CHAINels team has worked for over two years on the network, and already has some tools set up. To maximise compatibility, and to use their expertise on the tools, we have decided to use their tools as well. We will give a short description of these tools. Other suggested tools that may be relevant to the OAT will be described in the section 'Additional Tools'.

2.1 Languages

As the OAT will be a fully browser-based tool, web-based languages will be used. This involves markup languages, server- and client-side programming languages. The language used for accessing the database is often separate from the server-side programming languages, but at CHAINels, this is a plugin for PHP.

2.1.1 HTML and CSS

Hyper Text Markup Language (HTML) is the core language for the World Wide Web. From the late 2000s on, development for version five of HTML has been active. In this new version, a lot of changes have been made already since the previous version. These changes include new tags to structure a website, new elements, better support for dynamic web pages, as well as removing some deprecated elements. One of the elements, `<canvas>`, may be of particular interest. With `<canvas>`, one can create a container to draw graphics in. These graphics usually are created by scripting on the fly, i.e. using JavaScript. Because HTML5 is new, it is mostly only compatible with the latest versions of modern browsers. Because a requirement from CHAINels only stated compatibility with the last version of each browser (as their website is HTML5-based), this should not pose a problem.

Cascading Stlye Sheets (CSS) is a style sheet language, used for the look and formatting of documents written in markup language. With CSS, the content and the presentation of the document can be separated, providing more accessibility, increasing flexibility, and allowing multiple documents to share the same style. This property is required, as the HTML of the pages is generated in PHP pages.

2.1.2 PHP

PHP: Hypertext Processor (PHP), is widely used to create web applications. PHP is interpreted by the web server to create a resulting HTML page to be rendered by the browser. PHP can be used for object oriented programming. This allows for larger and more complex systems to be created, utilising advantages object oriented programming languages have in combination with a large database. CHAINels strictly applies object oriented programming in PHP.

2.1.3 JavaScript and Ajax

JavaScript has gained a lot of territory as a scripting language. It is executed client-side in the web browser and provides interaction with the user. Amongst other programming styles, it supports object oriented programming. Ajax is an expansion of JavaScript, that enables executing a new server request without reloading the entire page. This is very useful for creating dynamic web pages.

2.2 Databases

In this section we distinguish two types of database: relational databases and NoSQL databases. CHAINels makes use of Redis, a NoSQL database. These two types of databases differ in a number of ways. The Dutch website Tweakers [8] made a comparison between the two types.

2.2.1 Relational databases

Relational databases consist of schema's, a two-dimensional table with attributes on the columns and entities on the rows. These tables can be linked together by making complex queries on the database. While relational databases are flexible, they can create a high load on the server. Horizontal scaling, the adding of additional servers, is very difficult with relational databases [1]. Only vertical scaling, the improving of the hardware, is possible. Another possibility is to use the computer's memory for caching to increase performance.

Relational databases work with the 'ACID principle', assuring atomicity, consistency, isolation, and durability. These properties make the relational database reliable, but may lead to problems. To edit an entity, it needs to be locked, disallowing other processes to access the same entity.

2.2.2 NoSQL databases and Redis

NoSQL databases do not have the relational setup. A NoSQL database can have more than two dimensions. Depending on which database is used, joins between schemas are not or hardly supported. Implementing this functionality is required by the developer. Without the relations between schemas horizontal scaling works very well.

NoSQL databases are capable of working with large sets of data. Google's BigTable database, for example, is capable of handling petabytes of data. The Redis database is an in-memory database, and only uses a hard disk for backing up. This ensures a very high speed for both reading and writing operations, which is helpful when working with large datasets. These properties should ensure the high speed of the OAT.

2.3 Administrative tools

2.3.1 JIRA with Bonfire

JIRA is a project managing tool. It can be used to track bugs, issues, and the progress of the project. It can also integrate with the program code, and be used for reporting and analysis.

The Bonfire extension can be used with JIRA to easily track bugs and other errors in web pages. On any website, one can activate the tool and select an area of the site which contains the bug or error. The tool then saves a screenshot of this error, along with information like the used browser, operating system, and other information that might be needed to solve the bug. The bugs are automatically inserted into JIRA as an issue, where it can be integrated into the work process.

2.3.2 SmartSVN

SmartSVN is a subversion (SVN) client, used to track version of the code. With SVN tools multiple people can work together on the same files. Additionally, users can work on the files locally to change or add code, without the risk of introducing bugs to every other user. Modifications from distinct users can be combined in the end. This is called branching. Also, SVN enables going back to an older, correct version of the so-called branch.

2.3.3 Jenkins

Jenkins is an online continuous integration tool. When program code is committed to the SVN, Jenkins automatically runs all the tests written. These runs can be automated to run every night, for instance. After the run, a report is created with the results and, if anything went wrong, a detailed output to debug the program. Jenkins not only tests for errors, but also for code quality, and the comment ratio. Jenkins also extracts the documentation from the code and transforms this into a group of pages that link to each other, that is useful for understanding the structure of the code.

2.3.4 Selenium

Selenium is a tool that enables automated browser usage. CHAINels uses this to automatically test their website as it behaves in a browser. In this way, bugs can be found that are not found by automated tests of the code.

2.4 Additional Tools

The tools MapReduce and Bootstrap were also mentioned. We have looked at these tools, but after consulting the CHAINels team we decided not to use these tools. MapReduce is used to execute actions in parallel, but the Redis database is expected to be sufficiently fast to handle the big data. However, if it turns out that this is not the case, measures will be taken. Bootstrap can be used to quickly create mock-up user interfaces, but in our agile approach we opted for other, less technical solutions.

3 The existing framework

In this section, we first explain how the code is structured, and what techniques are used. Second, we explain how the data from the website is stored.

3.1 Examination of the website

The CHAINels website is built using mostly object oriented PHP and JavaScript. A PHP library is used for the structure of the page, which replaces actual writing of HTML code. JavaScript calls are used to create the dynamic content on the website. The website is created using *pagelets*, small parts of the website. These pagelets are sent to the user based on priority. This way the loading performance of the website in increased. This technique is called BigPipe, and is developed by Facebook [3] .

In the back-end of the website, the company class is one of the most important classes, representing any company that is registered on the website. Every page has an action controller defining what actions can be done on that page. Within each controller is a function to send the statistics to the database. Calls to the database can be collected into a single batch, which can then be sent to the database together, to increase efficiency.

3.2 A survey of the collected data

As mentioned before, the database system that CHAINels uses for storing data is Redis. This database can be managed by using a web interface called phpRedisAdmin. In this interface, the stored data can easily be examined.

We discovered that the amount of data that CHAINels stores is not that huge yet, and therefore still easy to understand. However, in Redis, data is stored much different compared to a relational database system like MySQL. Redis relies on key-value relationships. CHAINels saves a user's email address (which also functions as the username on the website) as a key, that refers to the details of this user as the value. The user details are stored in a hash table, where the keys are *pass*, *name*, *cid* (company id) and more, together with their corresponding values.

Currently, every company on the website is managed by exactly one user. The current database already supports more users for one company (possibly with different rights), but this is not implemented yet in the front-end. For each company, one hash table is stored, containing the company's details (*name*, *country*, *description* etc). In separate hash tables and sets for each company its chains (with other companies), admin IDs, messages, (a reference to) statistics and other details are stored.

The usage statistics are stored in a separate group of hash tables. One hash table contains the page views of one page. A page view contains which company and user visited the page and when.

4 Web Usage Mining

Web Usage Mining (WUM) is the process in which the behavioural patterns of users on a website are analysed. It consists of three stages: data collection and pre-processing, pattern discovery, and pattern analysis [5]. Together, it provides valuable insight in user behaviour, on which the developers of the website can respond. This is essential for the website of CHAINels, and is not possible without having gathered the companies interests. Therefore, it is an important process for us to study.

4.1 Data collection and pre-processing

In the first stage of WUM, all data that gives insight in user behaviour is collected. The raw data is often unusable for direct analysis, and therefore requires pre-processing. Pre-processing combines the loose data from log data into user transactions from unique user, representing their activity on the website. One of the most important types of data is the clickstream. This involves all clicks that users do on the website. This shows for example how people navigate from one page to another (following hyperlinks), which buttons they press or which images they click to enlarge. Other actions that can be tracked are how often users post messages on the website (if that is an existing utility), how often they visit a certain (type of) page and how long they stay on that page. Also, is it possible to record their IP address, the browser they are using, the website that guided them to the current page and much more [5].

4.2 Pattern discovery

When the data has been transformed to a standardised user transaction, it is possible to search for patterns. Imagine we have a vector for each user representing the visited pages as a 1, and the other (unvisited) pages as a 0. If we transpose these vectors, and place them beneath each other, we have a matrix that represents the page visits for each user. If we have another matrix that tells us for each page which features (presented as keywords or other comparable units) are present on that page, we can get insight on which features are apparently of interest for a certain user, by multiplying these two matrices [5].

4.3 Pattern analysis

In the last stage, the user transactions of different users can be compared. This gives insight in groups of users that perform similar behaviour. This information is especially useful in recommendation algorithms. If for example user X has shown interest in pages A, B and C, and user Y visits pages A and B, the website might recommend user Y to visit page C as well, because it is likely that this page will also be of interest for him (because that also applied to user X). Pattern analysis can also give insight to paths from random pages A to B that are used often. Also, if users often seem to visit page B when they have visited page A too ($A \rightarrow B$), but if the path is relatively long, it may be a good idea to place a direct hyperlink on page A to page B.

4.4 Pros and cons of web usage mining

For a company like CHAINels, it is essential to analyse the behaviour of their users. It can be discovered in which way the users use the website. If this is analysed well, the website can be improved to make it more intuitive, or desired features can be added, and unused features can be removed.

There are also aspects that websites have to take into account when logging user activity. One might argue that a website should not log everything, and consider this as an invasion of privacy, because a user has the right to be anonymous. By clustering different users based on matching behaviour, a website judges a user on the group it is in. This de-individualisation might lead to a biased view of the user. For example, not all users that have an interest in sports also have an interest in baseball, despite many other sports lovers do.

When a user's behaviour is used to form recommendations, it is very important that the activities of this particular user can not be tracked back from that recommendation. Therefore, the data should be anonymised. The main issue here is that users should not be afraid that their activity will be visible for everyone. That would result in a decrease of trust in the website.

4.5 Existing tools for web usage mining and statistics

There are several tools available for registering web usage. Most tools require an addition in the code of the website, or access to the database. They often provide a combination of web usage mining and (visual) statistics. It is not our plan to implement one of these tools, because we will process the data that is already logged in the database of CHAINels (with JavaScript and PHP). Also, CHAINels uses a custom framework, that makes extensive use of Ajax. This means that users can perform several actions before the page changes, and that makes a lot of tools useless, because they track page changes. Therefore, it is of much greater use for our project to develop our own tool, that is also integrated into their framework. Still, it is useful to discuss a few of the available tools, in order to obtain ideas of which kind of data can be analysed and how to process this data.

4.5.1 Google Analytics

Google Analytics (GA) is a web-based tool that shows extensive statistics of the visitors of a website. It requires the web master to place a block of JavaScript code into each page that the web master wants to obtain statistics from. This block does not provide a lot of intelligence. Instead, it contains a reference to a large JavaScript file on the website of GA. This file is not designed to be easy to read, so it is not of much use to examine. It is therefore more useful to analyse the GA dashboard and its functions, in order to obtain ideas for data visualisation.

GA shows most of the standard visitor data, like the total amount of visitors (per time unit), the amount of pageviews and the percentage of new visitors. Another feature that gives interesting insight of user behaviour is the Visitors Flow (see Figure 1). This shows how users navigate through the website and which pages they visit. This may also be useful for the website of CHAINels.

GA also has its limitations. First, its details are based on cookies places on the visitor's computer. If these are removed, the statistics will be distorted. Also, if users have disabled JavaScript in their browser, their visit will not be recorded at all.

4.5.2 Chartbeat

Chartbeat is a tool that is mainly focused on what visitors are doing on the very moment. An interesting feature of Chartbeat is that it can track real-time when and how far people scroll the page. Scrolling the page proves that the user is actively using the page, instead of being idle. If a user is reading the page for a long time, it may be a useful page for this person. A similar feature may be very useful for us to implement in our product.

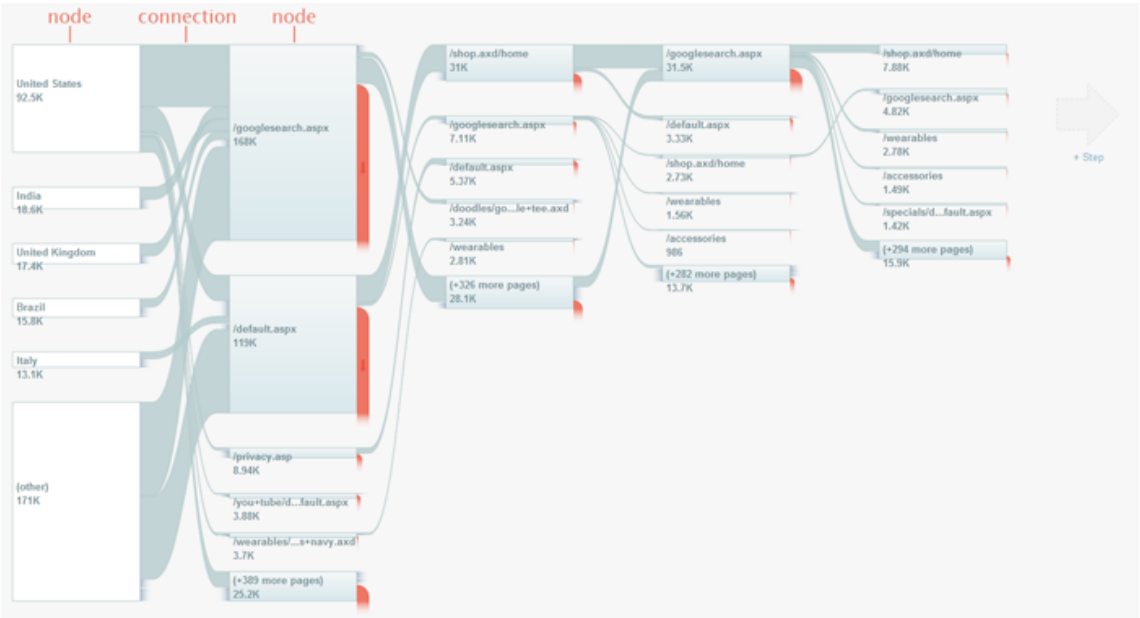

Figure 1: Google Analytics Visitors Flow

5 Efficient Design

To maximise efficiency of the OAT, a good user interface is crucial. In this section, we look at guidelines for creating an interface. In the second part of this section, we look at existing solutions to visualise the data that is collected by the CHAINels system.

5.1 Efficient User Interfaces

Since the beginning of computers, several ways of input have been available to users. Command line interfaces (CLI) have been one type that have been in use for a while. A CLI is efficient if you know how to use it, but novice users have difficulty using such an interface. With computers becoming more powerful, graphical user interfaces (GUIs) have become available with which almost any user can operate a computer. Research has been conducted as to what makes an interface good and useful.

In an extensive summarising article, Pascal Wiggers [12] starts with a description of the human side of the interaction. Users get input through sight and hearing, they produce output through touch. The senses taste and smell are not used (yet). When entering the human eye, light is eventually projected on the back of the eye, called the *retina*. This retina consists of rods and cones, with rods being sensitive to the intensity of light, and cones being used to detect colours. Additional special nerve cells exist on the retina called X-cells and Y-cells. X-cells are used to detect patters, where Y-cells detect movement. Human can see sharp in the centre of their focus, while the vision blurs in peripheral vision. Sound in interfaces is less effective, as interfaces who create noises all the time may simply become annoying.

Typing on a keyboard and moving peripherals are usually the two types of input a user can create. These physical movements take time, from experiments a result came, known as Fitt's Law:

$$T_m = a + b * log_2(\frac{d}{S} + 1) \tag{1}$$

, where a and b are empirical constants, d is the distance to a target, and S the size of the target.

Human memory is divided in short term and long term memory. Short term memory is fast but limited. On average people can remember 7±2 items. This capacity may be a limiting factor for speech based interfaces, but has less impact when all options can be presented on a screen so nothing has to be remembered. Long term memory is relatively slow but almost unlimited, while forgetting anything goes very slow. This is where humans store everything they know. Long term memory can be subdivided

into episodic memory and semantic memory. Episodic memory contains experiences and events, while semantic memory contains data and facts. When tasks are performed often, actions become systematic as long as it succeeds every time.

Users always have a mental model of how the program works. This initial idea varies per individual. The closer the actual behaviour of the system is to the mental model of its user, the more useful the system becomes.

As Wiggers also explains, interaction with a system consists of a cycle in which some steps are performed:

- Establish the goal
- Form the intention
- Execute the action
- Perceive the system state
- Interpret the system state
- Evaluate the system state with respect to the goals and intentions

This cycle describes having a goal, creating and executing an action to get to that goal, and after execution evaluating the result. Navigation in a design should be able to answer a couple of questions of the user: *Where am I?*, *What can I do?*, *What will happen?* and *Where have I been?* . The first question states that the state of the program should always be visible. The second can imply that every action a user can execute should be visible, while actions that are not available are not. The third question demands that icons and text on buttons should indicate what they are for, and the last question makes it easy for a user to get to an earlier point.

Wiggers concludes with a list of design guidelines:

- **Predictability** relates to the ease with which users can determine what the result will be of their actions.
- **Honesty** states the system should do as it promises.
- **Familiarity** is about the first impression of a system. If an user sees icons and elements he or she recognises, proper interaction is more likely.
- **Consistency** states the interface should be consistent in its appearance. Fonts, colours and other elements should remain the same throughout.
- With **recoverability** a user should be able to recover from errors easily.
- **Substitutivity** says it should be possible to switch between values for, for instance, input sizes, allowing both inches and centimetres.
- Different people have different wishes, **customisability** allows them to alter the interface to fit their needs and preferences.

Shneiderman's [10] list of golden rules supports the values of consistency, error handling, and easy reversal of actions but points to

- offering informative *feedback*
- enabling frequent users to use *shortcuts*
- designing dialogs to *yield closure*
- and *reducing short term memory load*

Giving feedback gives the user information of the state the system is in. Humans forget about actions once they feel it is finished, yielding closure gives users the opportunity to be done with one action and focus on the next. While reducing memory load and using a lot of shortcuts may seem inconsistent, the use of shortcuts is an example of actions which may become systematic, as Wiggers mentioned.

Norman [7] mentions seven principles to make difficult tasks simpler. In this principles he wants to make use of the knowledge users already possess, i.e. using the mental model. He also mentions making things visible, and getting the mappings right, which are consistent with Wiggers' guidelines of predictability and honesty.

Finally, some strategies on user centered design (UCD) have been devised. Abras [2] gives some steps to incorporate the user in designing a system, while Maguire [4] gives a more extensive framework to involve the user. The steps generally require more time and thus more money in the development phase, with feedback from the user on prototype interfaces, and an iterative development approach. However, they have benefits when the interface is in use because it requires less time to train with the interface, less errors are made by the users, and they have a higher efficiency.

We would like to mention the Windows User Experience Interaction Guidelines [1] which contains an extensive set of rules to create useful interfaces, covering almost any topic, with examples of correct and incorrect interface elements.

5.2 Efficiently displaying data

An effective interface is one part of the system, getting the message across in an acceptable manner is another. Several elements can be used to maximise efficiency, for example colour [6].

As said, human eyes perceive colour through the cones on the retina. These cones are all sensitive to one specific wavelength (and thus specific to one of the basic colours red, green, and blue), and sensitivity varies per wavelength. This sensitivity can degrade with age. Sensitivity is not equally distributed amongst the basic colours, with only two percent sensitive to blue, with almost one third of the cones sensitive to green, and the remainder sensitive to red. It is important to note the cones sensitive to blue are poor represented near the centre of the retina, resulting in a blue-blindness causing small blue objects to disappear when they are focused upon. It should be mentioned, about nine percent of all people are colourblind.

Murch concludes with a set of guidelines to use colour effectively:

1. Avoid the simultaneous display of highly saturated, spectrally extreme colours.
2. Pure blue should be avoided for text, thin lines, and small shapes.
3. Avoid adjacent colours that differ only in the amount of blue.
4. Older users need higher brightness levels to distinguish colours.
5. Colours change in appearance as the ambient light level changes.
6. The magnitude of a detectable change in colour varies across the spectrum.
7. It is difficult to focus upon edges created by colour alone.
8. Avoid red and green in the periphery of large-scale displays.
9. Opponent colours go well together.
10. For colour-deficient observers, avoid single-colour distinctions.

These guidelines help to make any interface more readable and prevent visual fatigue. With these guidelines one can create interfaces, without having running into trouble, which may be caused by varying sensitivity from individual users.

In a book by Shirley [9] , Munzner also defines some characteristics on the visualising of data sets. The visualisation provides a better way for humans to understand the given data set than plain text. Data can be distinguished into three categories: quantitative data which relates to numbers, categorical data which can be divided in categories, and ordered data which is ordered by some measure. Any object has some characteristics like area, length, or hue, and these characteristics have some characteristics of their own: distinguish-ability, separability, and pop-out. Certain object characteristics can be combined, like colour and shape, but combining characteristics like length and width can not, as they are automatically perceived by humans as area, which can not be easily perceived categorical.

Certain styles of visualisation or algorithms may be better with smaller data sets. Large data sets may be reduced to improve performance and efficiency. Text is still needed, in the form of labels and legends.

One way of visualising activity on a website is with a heatmap. A heatmap logs all the clicks the visitors of a site do on that site. This way it may easily become clear where people click to get to where they want to be. If a site has multiple ways to get to a certain point, it can become clear one may be obsolete. See figure 2 for a heatmap from the old Tweakers.net website.

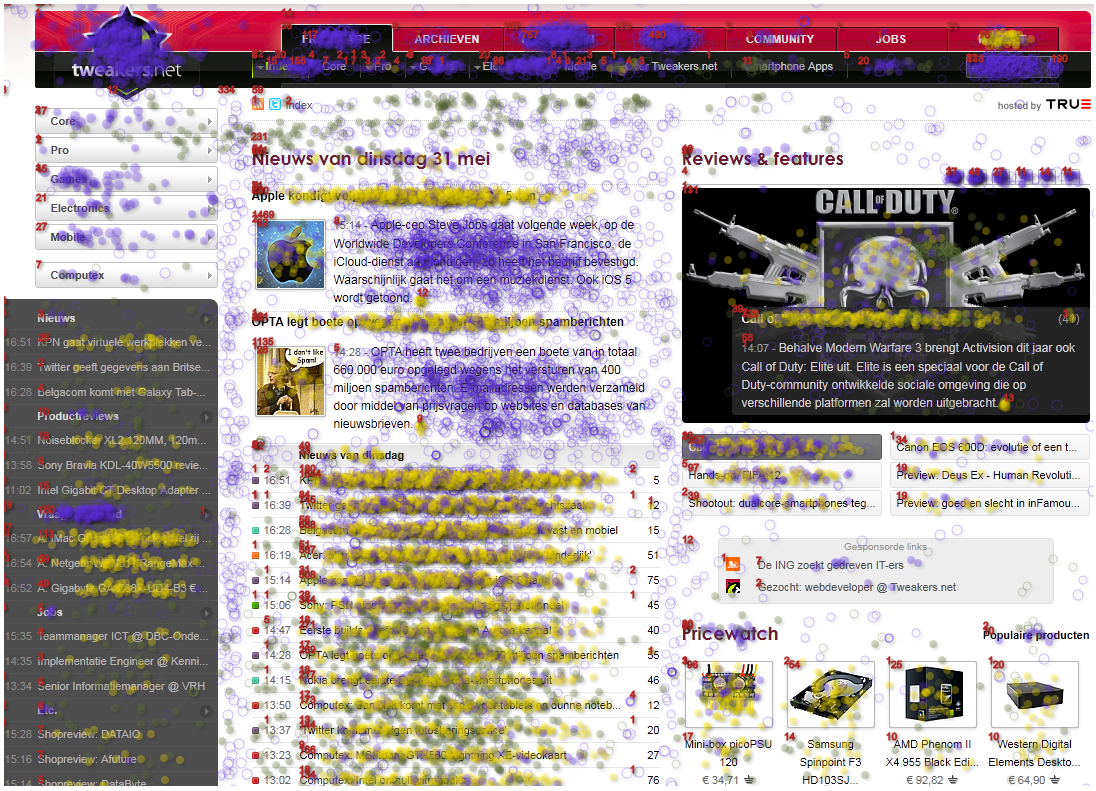

Figure 2: Old www.Tweakers.net site heatmap [11]

5.3 Survey of existing libraries for visualisation

As other libraries exist for our purpose, we take a look at these existing solutions, so we will not develop what already exists.

5.3.1 Highcharts

One fine example of a charting library is Highcharts (http://www.highcharts.com/). Highcharts has some points we can summarise:
Good:

- colourful visualisations
- wide browser compatibility
- dynamic visualisations

Bad:

- license fees required

While Highcharts is a good example of a useful library, and therefore we wanted to mention it, the license fees make it a highly undesirable option.

5.3.2 D3

D3 (http://d3js.org/) is a library which can be used to create a wide variation of charts and plots. The used language is JavaScript. Some characteristics are:
Good:

- loads of examples
- different kinds of visualisation available
- good looking, colourful visualisations
- open source

Bad:

- Not compatible with older versions of Internet Explorer

The d3 library gives numerous options to create visualisations from the data, which are also easy on the eyes. The fact that it is open source software makes it a possible option to use. Uncompatibility with older browsers is not an issue, as the OAT was only required to be compatible with the latest versions of popular browsers.

5.3.3 Dygraphs

Another class of libraries exist, from which Dygraphs (http://dygraphs.com/) is a good example. A lot of libraries exist with which charts and plots can be created. They have some characteristics:
Good:

- lightweight
- easy zoomable
- compatible with older browsers
- open source

Bad:

- only basic charts and plots possible
- plain looks

While Dygraphs may be open source, this library, along with others, has only functions to create basic charts and plots. As we may require other visualisations than basic charts, this option may not be the right one.

References

[1] Windows user experience interaction guidelines, September 2010.

[2] Chadia Abras, Diane Maloney-Krichmar, and Jenny Preece. User-centered design. *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*, 37(4):445–56, 2004.

[3] Changhao Jiang. Bigpipe: Pipelining web pages for high performance, June 2010.

[4] Martin Maguire. Methods to support human-centred design. *International journal of human-computer studies*, 55(4):587–634, 2001.

[5] Bamshad Mobasher. Web usage mining. *Web Data Mining: Exploring Hyperlinks, Contents and Usage Data*, pages 449–483, 2006.

[6] Gerald M Murch. Physiological principles for the effective use of color. *Computer Graphics and Applications, IEEE*, 4(11):48–55, 1984.

[7] D. A. Norman. *The Psychology of Everday Things*. Basic Books, New York, 1988.

[8] Joost Schellevis. Nosql - maar wat is het dan wél?, November 2011.

[9] P. Shirley and S.R. Marschner. *Fundamentals of computer graphics*. Ak Peters Series. A K Peters, Limited, 2009.

[10] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley, 1987.

[11] F. Taken. http://tweakers.net/reviews/2791/tweakers-7-wat-is-nieuw.html.

[12] Pascal Wiggers. Reader human-computer interaction 2009-2010. 2009.

Appendix E: Requirements Analysis Report

Requirements Analysis Report

Henk Kant - *4024400*
Daniël Mast - *4092457*

June 29, 2013

Contents

Introduction

This document discusses the requirements for the Online Administration Tool (OAT) that will be developed, and its environment. The requirements have been separated in different categories.

1 Functional requirements

The list of functional requirements which the OAT is expected to implement.

Overall statistics:

- The total number of unique visitors, chains, registered companies, credits, posts, and invites in the system.
- The total number of unique visitors, chains, registered companies, credits, posts, and invites over a time period[1] to be specified by the user, or via predetermined time periods[2].
- The number of gained credits (total or per time period) should be dividable per action with which they were gained, for both real time and total numbers throughout the OAT.
- The registered companies should be dividable in industry, number of employees, and region, for both real time and total numbers throughout the OAT.
- The number of chains should be dividable in sent requests and accepted requests throughout the OAT.
- The number of invites should be dividable in sent requests and accepted requests throughout the OAT.
- For both chains and invites, the average time between requests and accepts should be available.
- For both chain and invite requests, a percentage of accepted requests should be calculated.
- The number of posts should be dividable per type of user[3] throughout the system.
- In real-time the number of online[4] users (logged in and total) with logged in users dividable in region or industry. A real-time list of chains requests and accepts, invites sent and accepted, credits gained or spent, companies who joined, and posts made, during a period that is customisable by the user.
- A list of companies, based on activity[5], to show most and least "active" users.
- The average amount of recommends and comments on posts. The amounts should be dividable in type of message[6] and in company number of employees, region, and industry throughout the OAT.
- How companies which signed up got to know CHAINels.
- Charts showing the percentages of different options in user statistics[7].

Per company:

- The amount of time it took the company to complete the sign-up process.

[1] No minimum or maximum time limits exist.
[2] They may be logical, but to be determined, time periods, for example, 1 hour, 4 hours, 1 day, 1 week, etc.
[3] Currently only "Editor", "Manager" and "Spectator".
[4] The definition of online is to be determined.
[5] The function and the arguments to calculate the rank of the company are to be determined.
[6] Current types are "news", "event", "supply".
[7] As of writing consisting of browser, location, operation system, account and company id (if any), and referrer page.

- The phase (object CStatus) the company is in, and the percentage of fields (name, address, what services the company can offer, etc.) the company has filled out. The time they took to come get to that phase since sign-up, and in total time.
- The time they spent in the introduction (if any).
- The total amount of chains, chain requests, credits, sent invites, and posts of the company.
- The amount of chains, credits, received invites, sent invites, and posts of the company over a time period specified by the user[1], or via predetermined time periods[2].
- A rating to show how active[4] the company is.
- The pages a company views and the messages a company posts should be dividable in types of accounts
- The average amount of recommends and comments on a post.

Per page:

- A visitors flow, showing where the user came from and what the next page[8] the visitor goes to is (if any).

Per user:

- A list of statistics about the user[7].
- the amount of pages[8] the user visits per session and on average.

Management requirements:

- Users should be able to delete accounts of companies.
- The amount of credits which are rewarded for certain actions should be changeable.
- A notification to all users can be sent out to notify all companies at once.
- Companies' requests to be featured on the homepage should be handled through the OAT.
- Feedback from the companies should be viewable.
- Database access should be available.
- Error reports from the site should be viewable.
- Right management should be in place, to determine which user can view which parts of the OAT. New users should be easily added or removed. Rights of users should be editable.
- Notifications should always be visible to inform users about important events, like an error report.

Visualisation requirements:

- The tool should implement data currently available from the CHAINels site. Data from future improvement (and during the project) is not required to be included in the tool.
- The modules that show real-time data should by default be refreshed every minute. Users should be able to refresh this data manually, or set a specified interval, as long as this is in accordance with the resource requirements.
- Visualisations of the data should be zoomable where possible, to increase detail of visualisations on the fly.

[8] We define next page as a different URL.

2 Nonfunctional requirements

- The OAT should be intuitive. A long, extensive manual should not be necessary, even for users that are not highly skilled in computers or statistics.
- The graphs that will be used should be suitable for the data they represent. They should show a rich amount of data, but nothing irrelevant.

3 Environmental requirements

- The OAT should function in all modern version of the popular browsers: Firefox, Chrome, Safari and Internet Explorer. No major cross-browser differences should be experienced.
- The OAT should be accessible ONLY by CHAINels employees, and no one else. Therefore, authentication is required.

4 Architectural requirements

- Other programmers should be able to understand and edit the code in the future. This involves sound names for variables and classes, but also proper use of commenting.
- It should be possible to add extra modules/features in the future, without having to make a whole new system (modularity).
- It should be easy to add new natural languages in the future. This will be done via the current existing techniques in the CHAINels system.

5 Performance requirements

All requirements listed below should still apply when the web server is dealing with a high amount of traffic, both by visitors of the website as CHAINels employees accessing the OAT.

- The loading time of a random page (including graphical content) in the OAT should be responsive, on a wireless Internet connection in the CHAINels office. Responsiveness is tested via user tests.
- If permissible by the resource requirements, all data should be calculated when requested. If this is not possible, due to the growing size of the data, non-real-time data (that has to be processed before it can be shown) should be refreshed on some interval. The size of this interval will depend on the type of data. In order to deal with scalability, this time period may be changed when the website obtains more users in the future.
- Zooming in charts should be fluent, without visible stutter. The acceptance of this requirement is done through user testing.
- The represented data should be accurate, or not visible at all. No faulty data is permitted.

6 Design requirements

- The user should be able to drag the visible modules over the screen of the dashboard, to adjust the view to his or her liking. The users should also be able to remove modules that he or she finds irrelevant. It should always be possible to add them again later, in a separate settings page.
- The colour scheme used should allow users who are colourblind to effectively use the tool. Acceptance of the used colour scheme is determined by Vincent.

7 Resource requirements

- The back-end of the system should not make the CHAINels website remarkably slow, as they function on the same server and access the same database. The website may on average be no more than 10 percent slower with the OAT active than without.

Appendix F: SIG Feedback

[Analyse]

De code van het systeem scoort net 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Component Independence en Unit Size.

Voor Component Independence wordt er gekeken naar de hoeveelheid code die alleen intern binnen een component wordt gebruikt, oftewel de hoeveelheid code die niet aangeroepen wordt vanuit andere componenten. Hoe hoger het percentage code welke vanuit andere componenten wordt aangeroepen, des te groter de kans dat aanpassingen in een component propageren naar andere componenten, wat invloed kan hebben op toekomstige productiviteit. In dit geval valt met name op dat er cyclische afhankelijkheden bestaan tussen de componenten 'html' en 'controller' en tussen 'system' en 'controller'. Dit maakt het voor toekomstige ontwikkelaars lastig om uit te vinden waar een verandering plaats moet vinden, en om daarna de impact van de verandering in te schatten. Het is aan te raden om de afhankelijkheden tussen de top-level componenten nogmaals kritisch te bekijken en de cyclische afhankelijkheden op te lossen.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'getObjectsPerDayParam'-methode binnen de 'Statistics' class, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '// filter all data that fall between begin and end date' en '// determine the length of the given time period in seconds' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.