TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Informatics

IDK40LT

Akim Essen 120950

LINNWORKS INFORMATION DASHBOARD

Bachelor’s Thesis

Supervisor: Deniss Kumlander

PhD

Senior researcher

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Informaatikainstituut

IDK40LT

Akim Essen 120950

LINNWORKSI INFOPANEEL

Bakalaureusetööd

Juhendaja: Deniss Kumlander

PhD

Vanemteadur

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Akim Essen

18.05.2016

Abstract

The aim of this bachelor thesis is to develop a functional web application, which was requested by the LinnSystems. [1] LinnSystems is a company that develops and maintains a software, which helps customers around the world to optimize and automate their business in the field of electronic commerce.

Web Application will be called LinnDashboard. The aim of this application is to inform workers on the company's user growth, news, product reviews, the most used software modules by displaying this information using charts, grids and a map, and also to be customisable, so other developers could add their Views, which will benefit the company's productivity.

This Web Application will be shown on Smart TVs in every department room.

This thesis is written in english and is 62 pages long, including 5 chapters, 44 figures and 2 tables.

Annotatsioon

Bakalaureusetöö eesmärk on luua uued veebirakendused LinnSystems jaoks. LinnSystems on ettevõte, mis tegeleb tarkvara arendamisega, mis aitab kergendada inimeste äritegevust ja aitab neid müüda oma asju Internetis.

Lõputöö kirjutamisel loodud veebirakenduse nimi on LinnDashboard. Selline veebirakendus teavitab oma kasutajaid juhul kui klientide arv on suurenenud, teavitab jooksvatest uudistest, uutest toodetest ettevõttes, tähistab missugused tarkvara liidese osad on tihti kasutatavad. Kõik ees nimetatud informatsioon on näidatud diagrammides ja kaartide abil. Kõike on võimalik muuta ja lisada midagi uut kaasa.

Selleks, et see uus rakendus saaks töötada ettevõte peamise toode peal (Linnworks.Net), peavad olema lisatud mõned täiendavad funktsioonid. See versioon LinnDashboard lisarakendusest on algversioon ja selle loomine näitab, et on piisavalt ruumi võimalikuks edasi kasvamiseks. Rakendus on ka kohandatud selleks, et kõik teised ettevõtte arendajad saaksid temaga töötada ja teha täiendavaid muudatusi vajaduse korral. LinnDashboard on loodud selliseks, et iga selle kasutaja (ka siis kui ta ei tea koodi kirjutamise printsiipe) oleks võimeline kergelt lisada oma „View“ ja „Visualiseerimine“, kuid selleks temal peab olema ettekujutus sellest, kuidas rakendus töötab.

Käesoleva töö autor kirjutas selle rakenduse iseseisvalt ja tema poolt rakendus oli ka testitud. Seoses sellega, et lõputöö kirjutamisel arendatav tarkvara oli tehtud kindla firma jaoks, olid korraldatud kohtumised üks kord nädalas ettevõtte esindajaga. Kohtumiste jooksul näidati, kui palju oli tehtud ja konsulteerimise järgi tehti otsuseid, mida on vaja lisada või teha uuesti. Selline organiseerimine aitas projekti paremaks teha ja lõpuks saada kvaliteetse taotluse.

Ülalkirjeldatud veebirakendus on ette nähtud kasutamiseks Smart TV igas ruumis.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 62 leheküljel, 5 peatükki, 44 joonist, 2 tabelit.

List of abbreviations and terms

CEO	Chief executive officer
LinnSystems	E-commerce software company
View	Page of a dashboard
External View	View that is filled by an external web page
Visualisation	Visual element of a View
Kibana	Data visualisation tool
Monitis	Application performance monitoring tool
Module	One of a set of tools from LinnSystems software
Smart TV	Television with integrated internet
API	Application program interfacce
Logstash	Data collection engine
LinnDashboard	Information dashboard web application
C#	Programming language
JS	JavaScript programming language
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
TypeScript	Programming language developed by Microsoft
ASP .NET	Server-side web application framework

Web API	Template for building REST web applications
RESTful	Representational State Transfer. Software architectural style of World Wide Web
jQuery	Javascript library
AJAX	Asynchronous Javascript and XML.
GET	Web request
AngularJS	Web application framework
Mercurial	Version control system
IDE	Integrated development environment
ADMClient	Tool that staff uses in LinnSystems, to simplify tasks
Linnworks.net	Web e-commerce application that LinnSystems provides as a service
SQL	Structured Query Language
UI	User Interface
Embedded Resource	Visualisation, that is take from an other webpage.
Gamification	Application of game-design elements and game principles in non-game contexts

Table of contents

List of figures

List of tables

1. Introduction

To inform workers on the company's news, subscriptions growth, different software products popularity and plans for the future, we have a meeting two times a year, where the CEO makes a presentation. This is not the most efficient way of informing workers, because it is performed rarely and some of the workers can not be present, since several departments work in shifts. The aim of this meetings is to show people that the work they do plays a big part in the company's growth, connect different departments and motivate them.

For this, LinnSystems has requested the author of this thesis to develop a web application, which can constantly inform people in the company by showing useful information and also making this web application customizable, so other developers could work on it and add their additional Views.

1.1 Company's Requirements and Specification Development

1.1.1 Functional Requirements

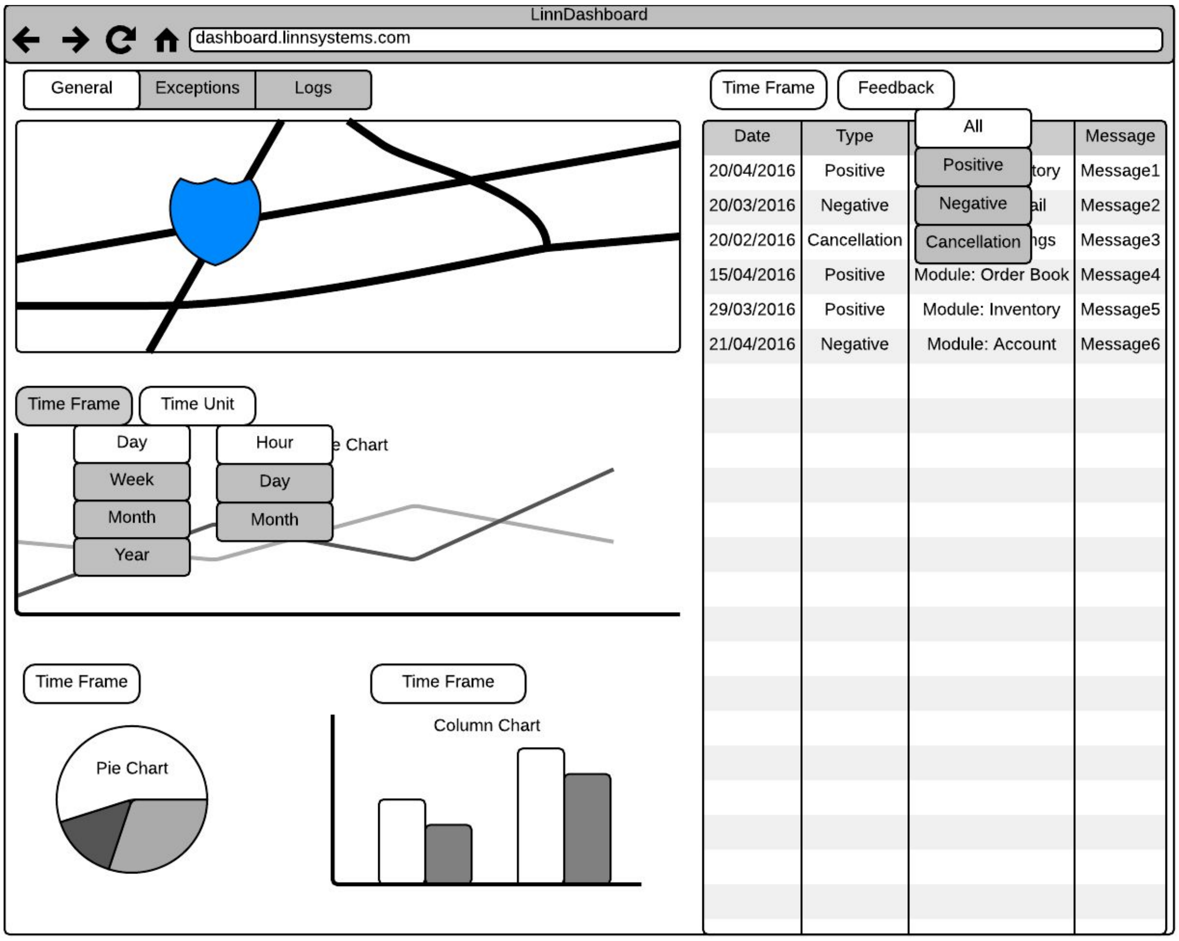
- Dashboard should consist of Views. Dashboard is able to hold several Views.
- Every View consist of Visualisations. Views are able to hold several Visualisations.
- Visualisations show information via charts or a grid.
- Developers will be able to create their own Views with Visualisations or insert a Visualisation into existing View with ease.
- Each Visualisation will have custom width, height and position, which developers can set manually.
- Views need to switch between each other every N seconds, minutes or hours, which can be set by the developer. If needed, View can be locked, so it will not switch to the next View. Views need to be refreshed manually if needed and, also, user will be able to switch to another View manually.
- Views need to be refreshed every N seconds, minutes or hours. Every View need to have their own refresh time that will be set by the developer.

- User should be able to filter the information, which is shown by the Visualisation. Developer will be able to add his own custom Filter, if needed.
- Developers will be able to add a custom View (External View), which hold the information that needs to use a third-party dashboard.
- There will be 3 Views by default. One General View, which will hold ‘User Feedback’, ‘Users online’, ‘Frequently used Modules’, ‘Feedback statistics’, ‘Users online map’. Other Views are External Views: Kibana and Monitis . [2] [3]
- Later there should be a View for each of the company's departments.

1.1.2 Non-Functional Requirements

- Developers should easily understand and use the web application.
- Dashboard needs to work on a Smart TV.
- Dashboard will only be available on the company intranet.
- Information needs to be understandable, so everyone in the company will know what they are looking at.
- Views need to work without any delays, in other words Visualisations need to appear instantly after each refresh.

1.1.3 Interface mockup



Date	Type		Message
20/04/2016	Positive	tory	Message1
20/03/2016	Negative	ail	Message2
20/02/2016	Cancellation	ngs	Message3
15/04/2016	Positive	Module: Order Book	Message4
29/03/2016	Positive	Module: Inventory	Message5
21/04/2016	Negative	Module: Account	Message6

Figure 1. Dashboard mockup

General View will have 5 Visualisations: Grid, Map, Line Chart, Pie Chart, Bar Chart.
Each Visualisation can have any amount of filters.

Grid will show the Feedback that users leave on linnworks.net about our company products. It has 4 columns: Date, Type, Info and Message. Type will show what kind of Feedback it is: Positive, Negative or Cancellation. Info contains our product module name, that the client selected for Feedback or additional information that was left by our Sales Department. Message contains the feedback message the client left. Grid has two Filters: Time Frame and Feedback. Time Frame will allow users to select the time range of the feedback (last days, weeks, months or even years).

Map shows how many customers are subscribed to our product in each country. If there are clients in a particular city, then the city will be marked by colour shade. Shade intensity will show how many customers are located in a city. Map will not have any Filters.

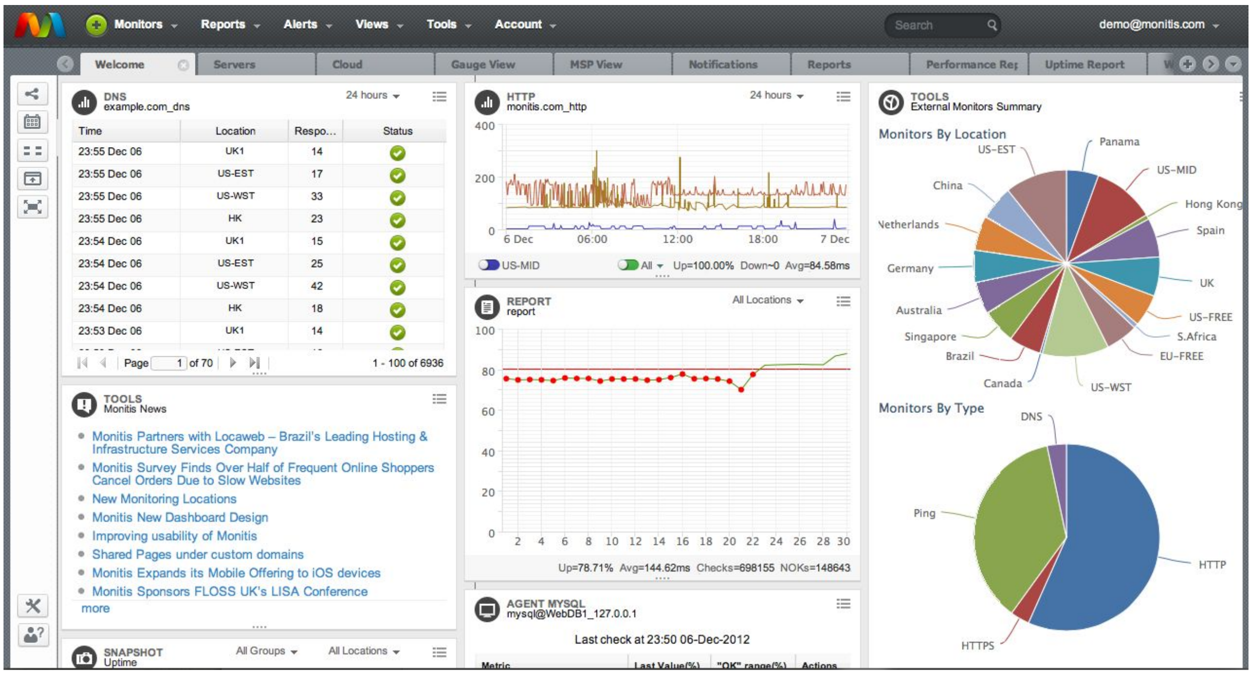
Line Chart shows how many customers were online in a particular time frame. X axis shows the time frame and Y axis show the number of online users. Line Chart will have two Filters: Time Frame and Time Unit. Time Frame, like in the Grid, will show how many users were online in the last day, week, month or year. Time Unit changes the unit of the Y axis. Time Unit has 3 units: Hour, Day, Month.

Pie Chart shows the numerical proportion of the Feedback types that our product receive. The chart will be divided into 3 slices: Positive, Negative, Cancellation. It will have only one Filter, Time Frame. Like in the charts above, it will show the feedbacks numerical proportion in the last day, week, month or year.

Bar Chart shows the proportion, for each module in our product, between Positive and Negative feedback. It will only have one Filter, Time Frame. It will show the feedback that was received last day, week, month or year.

1.2 Overview of existing applications for information dashboards

Author of this thesis decided to find an existing web application that could do the necessary requirements that were given by the company. Below is a brief description of the web Applications that were found:

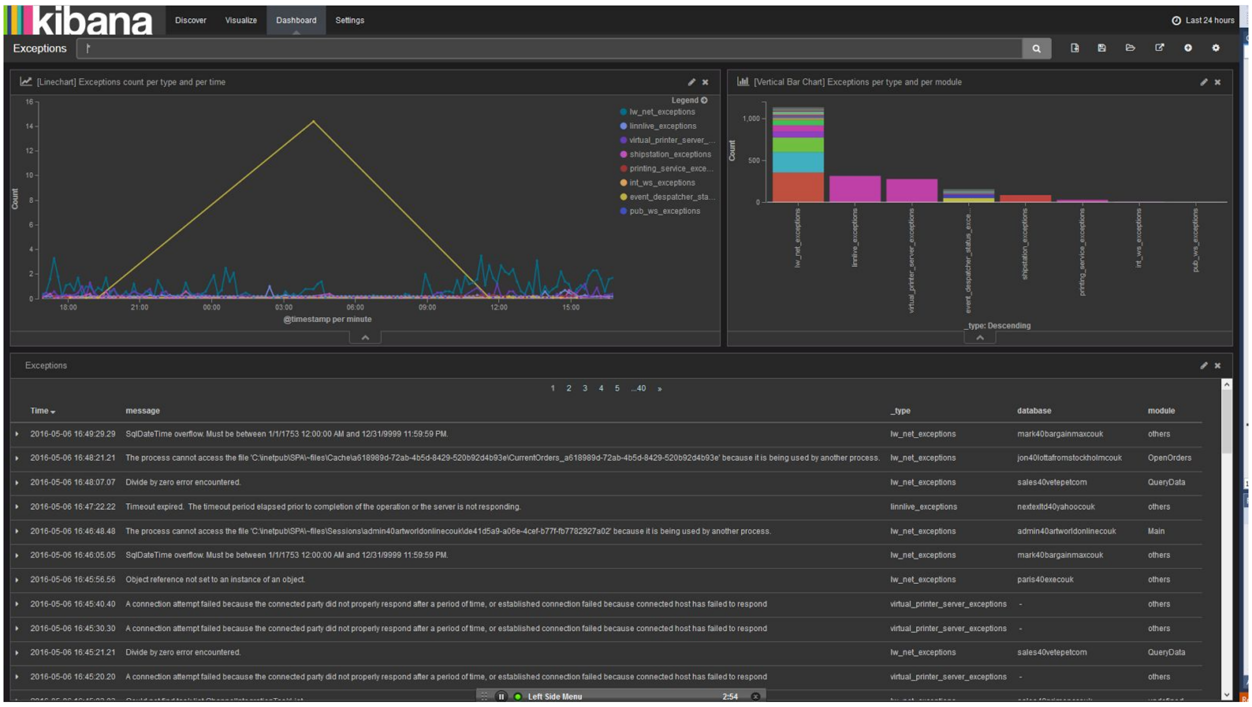


Time	Location	Respo...	Status
23:55 Dec 06	UK1	14	✓
23:55 Dec 06	US-EST	17	✓
23:55 Dec 06	US-WST	33	✓
23:55 Dec 06	HK	23	✓
23:54 Dec 06	UK1	15	✓
23:54 Dec 06	US-EST	25	✓
23:54 Dec 06	US-WST	42	✓
23:54 Dec 06	HK	18	✓
23:53 Dec 06	UK1	14	✓

- Monitis Partners with Locaweb – Brazil's Leading Hosting & Infrastructure Services Company
- Monitis Survey Finds Over Half of Frequent Online Shoppers Cancel Orders Due to Slow Websites
- New Monitoring Locations
- Monitis New Dashboard Design
- Improving usability of Monitis
- Shared Pages under custom domains
- Monitis Expands its Mobile Offering to iOS devices
- Monitis Sponsors FLOSS UK's LISA Conference

Figure 2. Monitis screen

Monitis - web application, which monitors the websites and servers uptime, full page load and transaction monitoring. This application does not meet the company's full requirements, however its API will be used as an External View, which will show the status of our servers. [4] This will help developers and support to acknowledge when the server is down, without going to the IT Administrator every time, which will help teams to work more efficiently.



Time	message	_type	database	module
2016-05-06 16:49:29.29	SqlDateTime overflow. Must be between 1/1/1753 12:00:00 AM and 12/31/9999 11:59:59 PM.	lw_net_exceptions	mark40bargainmaxcouk	others
2016-05-06 16:48:21.21	The process cannot access the file 'C:\inetpub\SPA\~files\Cache\a618989d-72ab-4b5d-8429-520b92d4b93e\CurrentOrders_a618989d-72ab-4b5d-8429-520b92d4b93e' because it is being used by another process.	lw_net_exceptions	jon40lottafromstockholmcouk	OpenOrders
2016-05-06 16:48:07.07	Divide by zero error encountered.	lw_net_exceptions	sales40vetepetcom	QueryData
2016-05-06 16:47:22.22	Timeout expired. The timeout period elapsed prior to completion of the operation or the server is not responding.	linnlive_exceptions	nextexltd40yahoocouk	others
2016-05-06 16:46:48.48	The process cannot access the file 'C:\inetpub\SPA\~files\Sessions\admin40artworldonlinecouk\de41d5a9-a06e-4cef-b77f-fb7782927a02' because it is being used by another process.	lw_net_exceptions	admin40artworldonlinecouk	Main
2016-05-06 16:46:05.05	SqlDateTime overflow. Must be between 1/1/1753 12:00:00 AM and 12/31/9999 11:59:59 PM.	lw_net_exceptions	mark40bargainmaxcouk	others
2016-05-06 16:45:56.56	Object reference not set to an instance of an object.	lw_net_exceptions	paris40execouk	others
2016-05-06 16:45:40.40	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond	virtual_printer_server_exceptions	-	others
2016-05-06 16:45:30.30	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond	virtual_printer_server_exceptions	-	others
2016-05-06 16:45:21.21	Divide by zero error encountered.	lw_net_exceptions	sales40vetepetcom	QueryData
2016-05-06 16:45:20.20	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond	virtual_printer_server_exceptions	-	others

Figure 3. Kibana screen

Kibana - web application, interface to data exploration, which helps to display a vast amount of data. It is usually used with events type date (timestamps), for example system logs. For this to work, there is a need for many additional plugins, which will convert all the necessary data to the needed format, but it will not work for some of the data needed by the company, for example news. It was decided to use Kibana as an External View with **Logstash** , which helps to get logs from our software products and then using Kibana to transform this information to different charts. [5] This will help developers to acknowledge any error or issue the client came across and deal with it as soon as possible.

This additional web applications will be used to enrich the LinnDashboard and will make the workflow more efficient.

2. Choice of tools

2.1 Programming languages

The main reason behind choosing which programming language to use for the backend was which language is used in our company, so other developers can freely work with the dashboard. Because of that C# was selected. The dashboard needs to work on a Smart TV and it was decided that dashboard needs to be developed as a web application, so it was a great plus that C# is one of the leading Web languages and is easy to read and develop.

Frontend was developed with JavaScript, HTML as a markup language and CSS as a style sheet language. It was considered using TypeScript, because it provides type safety, classes and becomes very popular, also it is being developed by Microsoft, so the tools that are used to write in C#, work great with TypeScript. [6] However, TypeScript is not used in our company and the dashboard codes needs to be easily accessible by other developers. Also, TypeScript is designed for development of large applications and the dashboard is not in that scale to use this language. [7]

2.2 Frameworks and Libraries

Because the language for the backend is C#, it was decided to use **ASP .NET** framework and use the **Web API** template it offers, to build a RESTful application. [8] ASP .NET allows to create web services and applications that have the support for dynamic as well as static content. It uses C#, which is a object-oriented programming language, the framework will be easy to work with, because it helps to eliminate unnecessary amount of code and involves less coding for the developers.

For the Frontend part of the web application, it was decided to use a technology that we use in our company - **jQuery.** [9] jQuery has an AJAX support, which will enable the frontend of the web application to get necessary information via the GET call. [10] It is easy to use, has a wide open source community and enables developers to use less lines of code to achieve the same feature in comparison. While searching for a JavaScript library, which could fulfill all of the functional requirements of the web application, it was considered that jQuery is the best option out there and our developers are well accustomed to this library. It was considered using AngularJS, because lets developers to create Directives, Modules and etc, which could help to make the application more scalable. [11] However, because jQuery is lighter than AngularJS, has all of the necessary functions, and it is also not wise to add AngularJS for scalability in a small project, AngularJS will not be used in this project. [12]

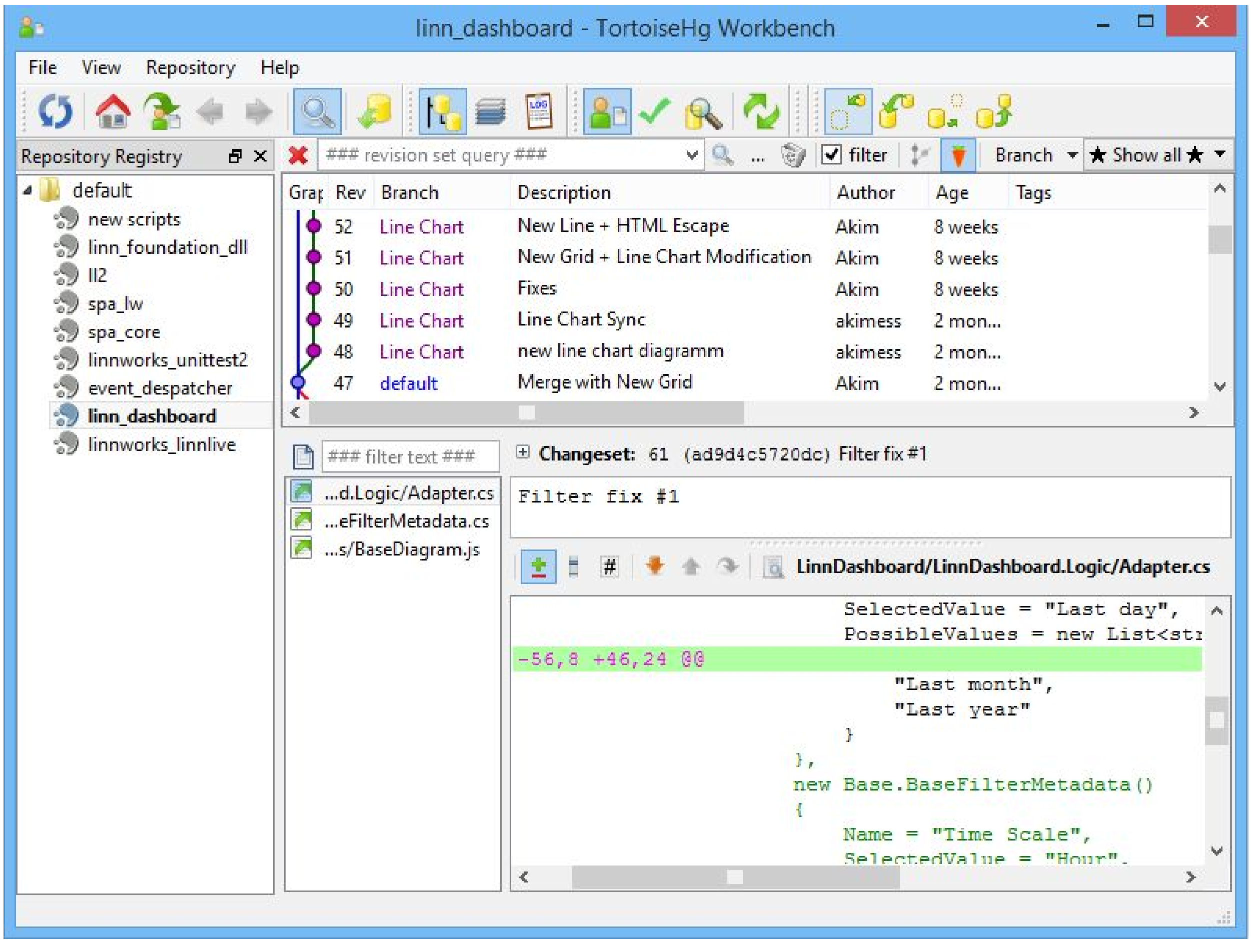
While searching for the javascript library, that will produce necessary Visualisation, author came across a large quantity of them. To decide which one is better for the application, it was necessary to test out some of them and see which could achieve functional requirements and with less coding. It is needed to use as few javascript libraries for charts and grid as possible.
The first one that caught authors attention was **Chart.js.** [13] Charts that were produced by this library were visually great, but there were a few problems when refreshing them. When refreshing, the new chart was put on top of an old one and when hovering with the mouse cursor over them, then they would switch constantly between each other. Workaround for this was to remove the old chart and draw another one, but if there is a lot of information needs to be drawn, then it would take a few seconds until it will appear after a refresh. Because of this author decided to try another chart library.
Another chart that the author came across was **Highcharts**. [14] However it was not lightweight and the charts were produced by creating an image on the charts library server, so it took a lot of time after refreshing the chart to draw a new one. Highcharts did not meet functional and nonfunctional requirements.
The last javascript library that was tested and then selected for use, was **Google Charts.** [15] It has many kinds of charts. The library is not lightweight, but the developer can select which charts he needs to import into the system. Charts are drawing and refreshing instantly. The customizability is easy to use and takes less coding than other chart libraries to accomplish. This library meets the functional and nonfunctional requirements, but the Grid in it is not great for our needs, because in the Grid we need to use Formatters. Formatters function is to add custom styles to any number of cells. While Google Charts Grid had this function, it was not adding custom styles to individual cells, instead it was using it for each column. Because

of this, author kept the Google Charts library for other charts, but there is still a need to find a better variant for the Grid.

First Grid library that caught authors eye was **SlickGrid.** [16] It was highly customisable and lightweight. However, by being highly customisable, it was not easy to use. Formatters took a lot of code to implement. Also, the author of this library stopped working on it in 2014. There were some bugs in it, which, of course, were fixed by other users of this charts on their repositories, but downloading each fix would be a lot of work and it was decided to continue searching for another grid.
Later author found not a javascript library, but a jQuery plugin that is used for drawing Grids. **jsGrid** is a lightweight client-side data grid control based on jQuery. [17] It supports basic grid operations like inserting, filtering, editing, deleting, paging and sorting. jsGrid is flexible and allows to customize its appearance and components. Formatters are very easy to use and can be set for cells individually by filtering. This plugin worked fast and the information was displayed correctly with custom stylized cells. In the end of testing, author made a choice to use this plugin for the Grid Visualisation.

2.3 Third-party tools

The development workflow needs to be managed and easily accessed on several computers. Author of this work used **Mercurial** version control system and **TortoiseHg** or **SourceTree** as a visual repository management tool. [18][19][20] Version control system helped clean the development workflow clean, by creating individual branches for each new functional addition to the web application. This was done for the purpose of keeping a working version on the default branch, while making functional additions and modifications on a separate branches, so after completing the addition they could be merged together and after that the separate branch would be closed. Author worked on different computers, one at work and one at home, so it was pretty easy to keep the work up to date.


linn_dashboard - TortoiseHg Workbench
File View Repository Help
Repository Registry
### revision set query ###
filter
Branch
Show all

default
- new scripts
- linn_foundation_dll
- ll2
- spa_lw
- spa_core
- linnworks_unittest2
- event_despatcher
- **linn_dashboard**
- linnworks_linnlive

Grap	Rev	Branch	Description	Author	Age	Tags
	52	Line Chart	New Line + HTML Escape	Akim	8 weeks	
	51	Line Chart	New Grid + Line Chart Modification	Akim	8 weeks	
	50	Line Chart	Fixes	Akim	8 weeks	
	49	Line Chart	Line Chart Sync	akimess	2 mon...	
	48	Line Chart	new line chart diagramm	akimess	2 mon...	
	47	default	Merge with New Grid	Akim	2 mon...	

filter text

- ...d.Logic/Adapter.cs
- ...eFilterMetadata.cs
- ...s/BaseDiagram.js

Changeset: 61 (ad9d4c5720dc) Filter fix #1

```
Filter fix #1
```

LinnDashboard/LinnDashboard.Logic/Adapter.cs

```
                 SelectedValue = "Last day",
                 PossibleValues = new List<str
-56,8 +46,24 @@
                     "Last month",
                     "Last year"
                 }
             },
             new Base.BaseFilterMetadata()
             {
                 Name = "Time Scale",
                 SelectedValue = "Hour",
```

Figure 4. TortoiseHg screen

As an IDE **Visual Studio** was used for backend and frontend. There was not much of a choice for using a different IDE with a compiler, because there is not a good .NET IDE on the market at the moment. Author of this thesis worked on a MacBook sometimes so it was difficult to find a .NET IDE. There was **Xamarin ,** but it was not free at the moment of developing this web application, but there was an Early Access Program for an application called **Project Rider** from JetBrains. [21][22] It was free for use, but it was not able to provide a code editor for the frontend part of the application. Because of this, **Visual Studio Code** was used as a code editor for frontend and sometimes for the backend code, but only for quick fixes, because it did not had a .NET compiler built in.

3. Preparation

3.1 Feedback

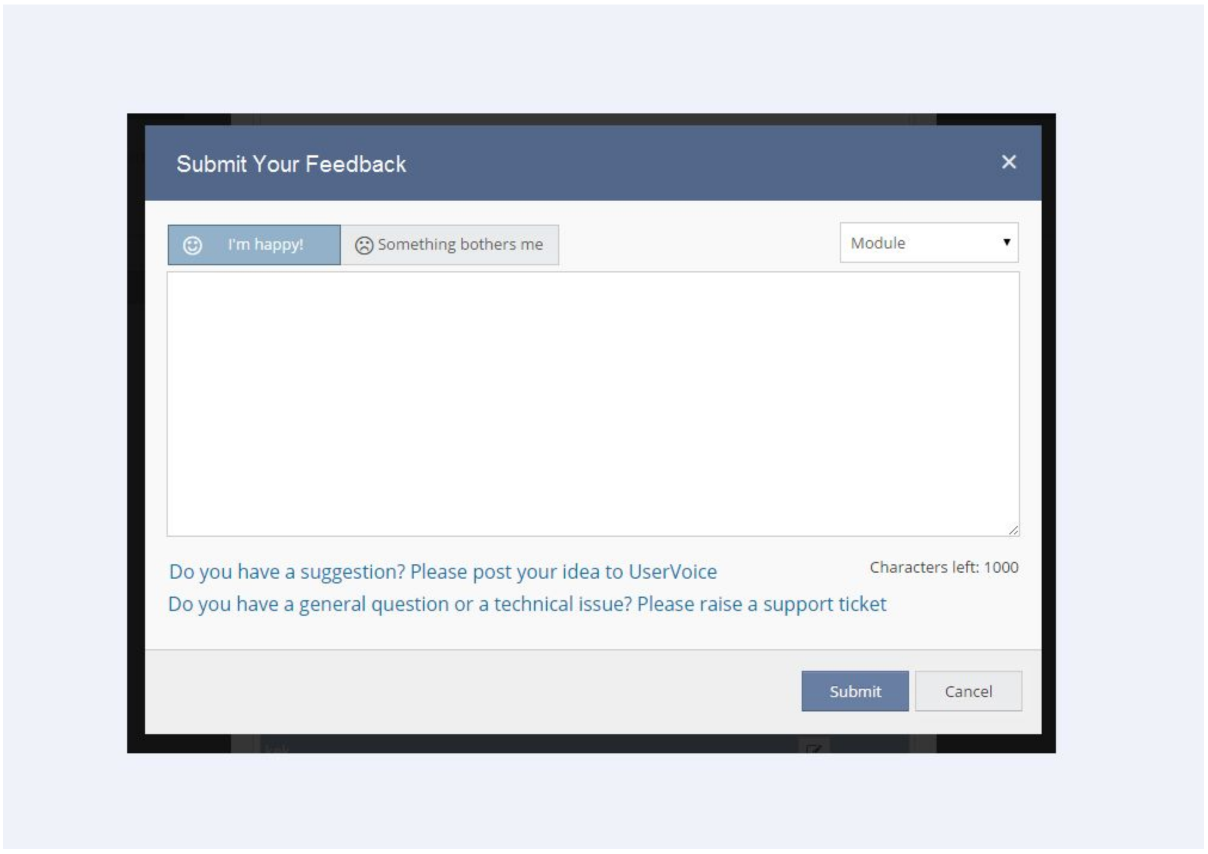
The purpose of Feedback functionality is to allow customers to share their neutral feedback or to leave a praise quickly and easily in a form of a short message - up to 1000 characters. Feedback window is not designed for discussion of issues or dialogues with our staff.

3.1.1 Visual implementation

The idea is to make a Feedback window accessible in two instances : firstly, for all customers in Linnworks.net, secondly, for Linnworks representatives inside ADMClient in Customers tab.
Below only the first instance will be described.

Linnworks.net will have a header panel, which holds different buttons, one of which is 'Feedback'. Once the button is clicked, it will open a pop up dialogue window, where an user can enter their feedback message and select what kind of feedback it is: Positive or Negative.

Upon clicking the button the next dialog window will appear.



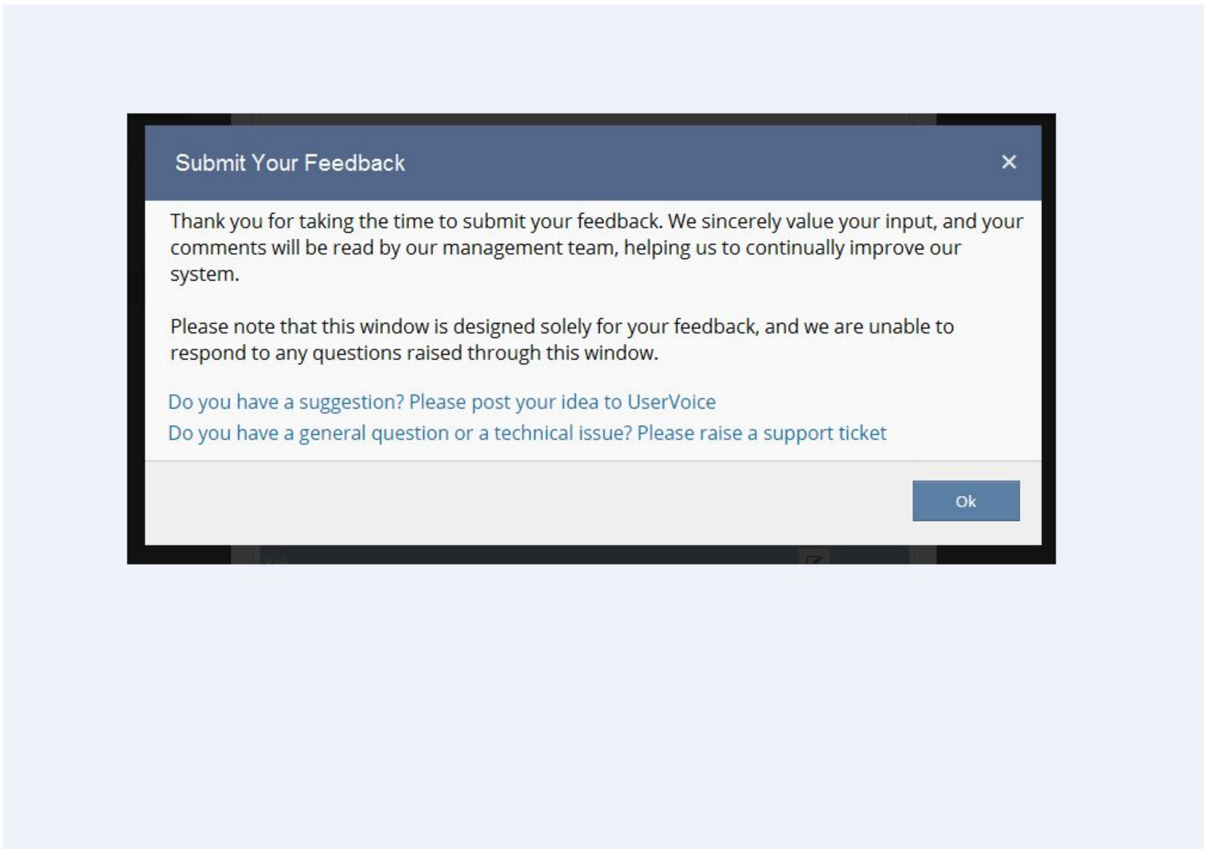
Do you have a suggestion? Please post your idea to UserVoice

Characters left: 1000

Do you have a general question or a technical issue? Please raise a support ticket

Figure 5. Feedback window in Linnworks.net

Upon clicking the Submit button, customer should get the following view in the same window. It lets customer know that we value their feedback and additionally gives a hint that this window is not designed for questions, they will not be answered here.


Submit Your Feedback

Thank you for taking the time to submit your feedback. We sincerely value your input, and your comments will be read by our management team, helping us to continually improve our system.

Please note that this window is designed solely for your feedback, and we are unable to respond to any questions raised through this window.

Do you have a suggestion? Please post your idea to UserVoice
Do you have a general question or a technical issue? Please raise a support ticket

Figure 6. Feedback window after Feedback submit.

3.1.2 Feedback database

Table 1. Customer Feedback Register database table

dbo.customer_feedback_register		
Column Name	**Column Type**	**Description**
pkRowId	bigint	Incremental row id
sid_registration	uniqueidentifier	Sid_registration to out which database it is
UserName	varchar(100)	Main database email or email of user who left a feedback
Staffid	Uniqueidentifier, nullable	Will be filled if feedback was added by someone from ADMClient
Time	datetime	Time

Module	varchar(50)	Module name
fkFeedbackTypeId	byte	1, 2 or 3
Message	nvarchar(1000)	Feedback message itself

Table 2. Customer Feedback Type Register database table

dbo.customer_feedback_type_register		
Column Name	**Column Type**	**Description**
pkFeedbackTypeId	byte	1 - POSITIVE 2 - NEGATIVE 3 - CANCELLATION
Name	varchar(50)	Name of a feedback type

- Sid_registration is there as a unique identifier indicating for which database this feedback is.
- Username will be picked either from account email address if the feedback is left by the admin of the system or from currently used email of the user inside this database.
- Staffid will be filled if feedback was added by someone from ADMClient. It is sometimes needed, because if the client cancels their subscription and did not mention why and someone from the Support or Sales Department knows the reason behind his cancellation, then he or she will be able to add the feedback themselves, so the information could be accessible by everyone in the company.

4. Design and Realisation

4.1 Design

The web application will be split into 3 projects: Logic, API (Backend) and Site (Frontend). Client side will request Dashboards data, the request will be sent to the Backend of the web application, there the API endpoint will catch the request and call a function from the LinnDashboard.Logic, which transfers the data to the correct format and gathers it from the third party resources. There are at the moment 3 Third Party Resources that will be used: Services, Database and Websites. Some of our databases are private and the information from them can only be requested from a service, for example, the information on how many users are online is located on a private database. Other information, for example, Feedback is located on database, that can be accessed through a simple SQL connection.

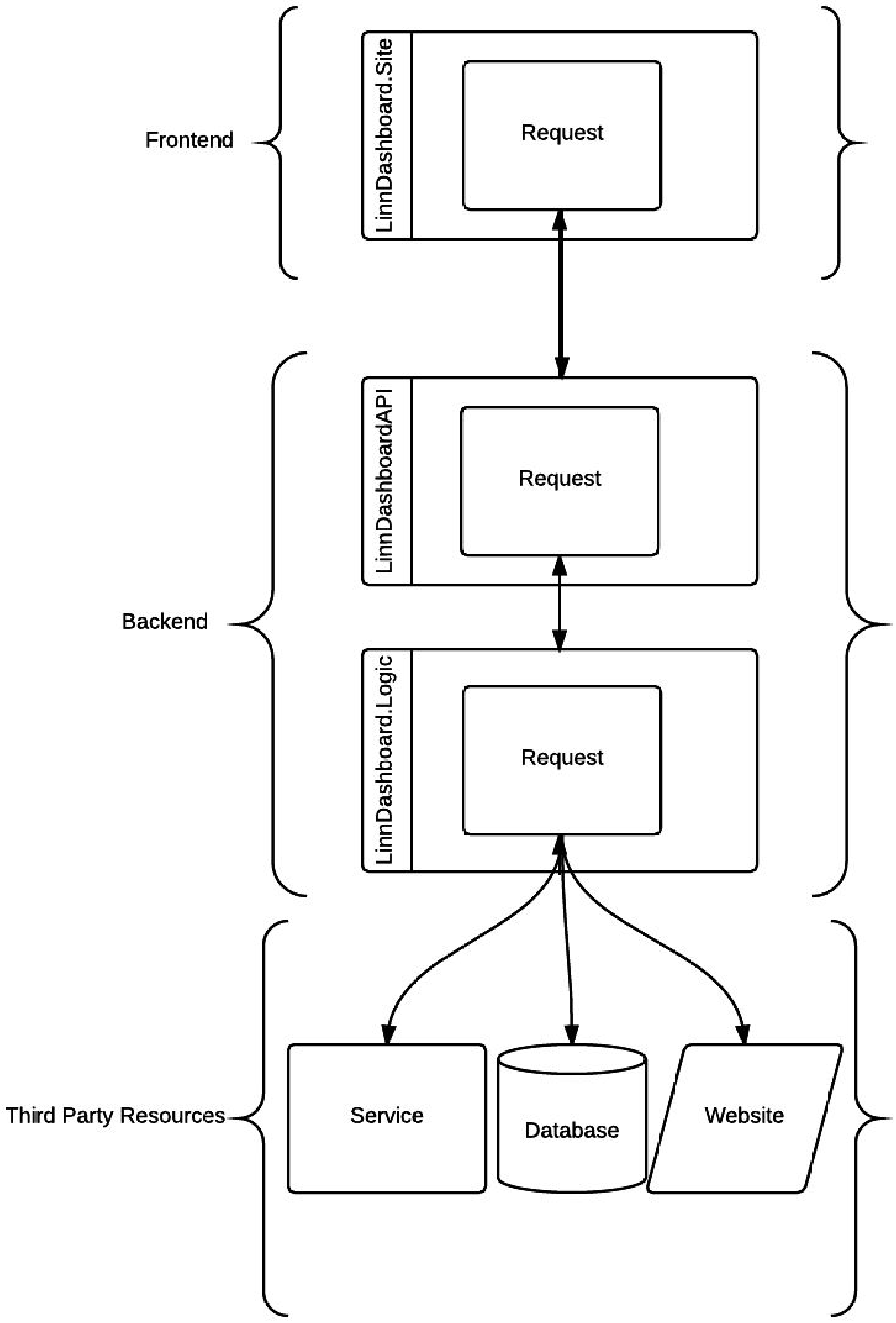


Figure 7. Basic Workflow Diagram

At the LinnDashboard.Logic information is gathered through the Adapter. Adapter creates Dashboards and fills them with Visualisations. Visualisations gather necessary data from Third Party Resources using DataSource classes and transform the information to the needed Diagram class, also they use Filters for Diagrams if needed. After which, the information is sent back to the API. Then the API endpoint sends it back to the client side.

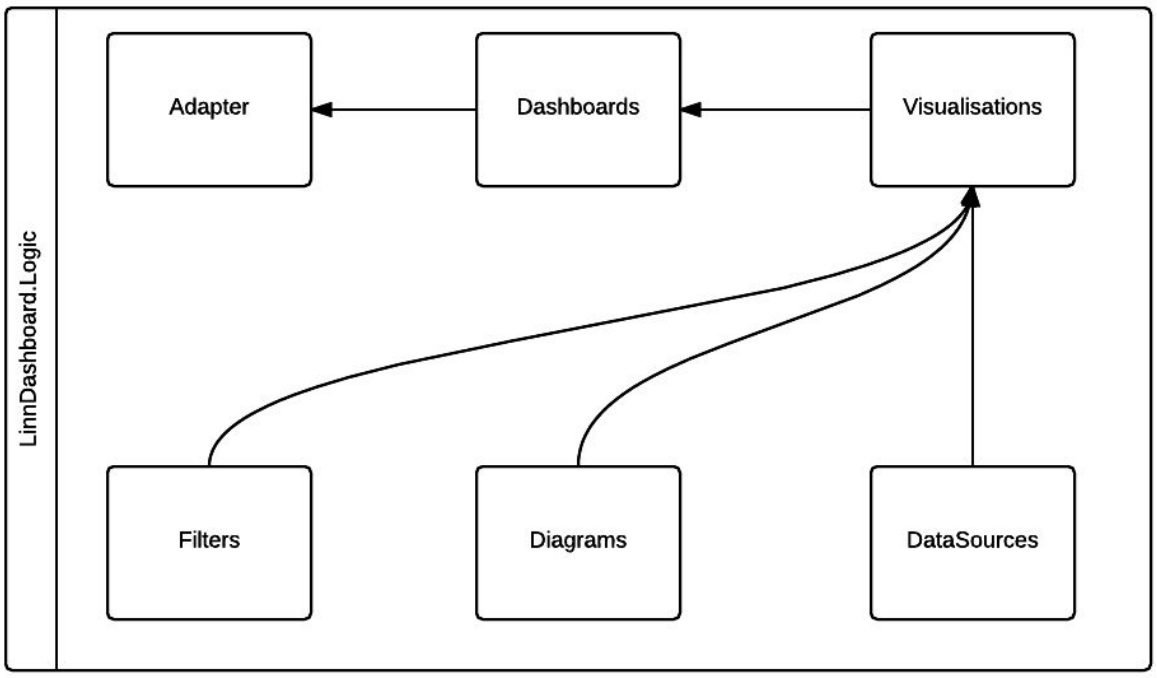


Figure 8. LinnDashboard.Logic workflow

4.2 Base Classes

Base classes are used to derive other classes. A base class is abstract and does not inherit from any other class and is considered parent of a derived class. Class derived from the base class inherits both data and behaviour. Web Application has 6 Base classes: BaseDashboard, BaseDataSource, BaseDiagram, BaseFilter (and also BaseFilterMetadata), BaseVisualization.

BaseDashboard class is used as a model for dashboards.

```
public class BaseDashboard
    {
        public Guid Id = Guid.NewGuid();
        public string Name;
        public List<Base.BaseVisualization> Visualizations;
    }
```

Figure 9. BaseDashboard class

BaseDashboard class consists of Id, Name and Visualizations. Id is a Guid and is used to identify the dashboard. This Id is later used in the client side of the web application to distinguish Views windows between each other. Name parameter is used in the UI, so the user could switch between the Dashboards/Views. Visualizations is a list of BaseVisualizations class. BaseDashboard does not have any child classes.

```
public class BaseVisualization
    {
        public Guid Id = Guid.NewGuid();

        public Base.BaseDataSource DataSource;
        public Base.BaseDiagram Diagram;

        public int Left;
        public int Top;

        public int Width;
        public int Height;

        public int Timeout;
    }
```

Figure 10. BaseVisualization class

BaseVisualization class also does not have any child classes and is used in the BaseDashboard class. Id parameter, like in the BaseDashboard class, is used to identify the Visualization. DataSource is a parameter, which contains the data for this particular Visualisation. Diagram is used to identify which Diagram is represented. Left and Top are used to position the Visualisation on the page. Width and Height are the dimensions of the container, in which the Diagram will be placed. Timeout is used for synchronisation, it specifies the time frame after which the Visualisation needs to be refreshed.

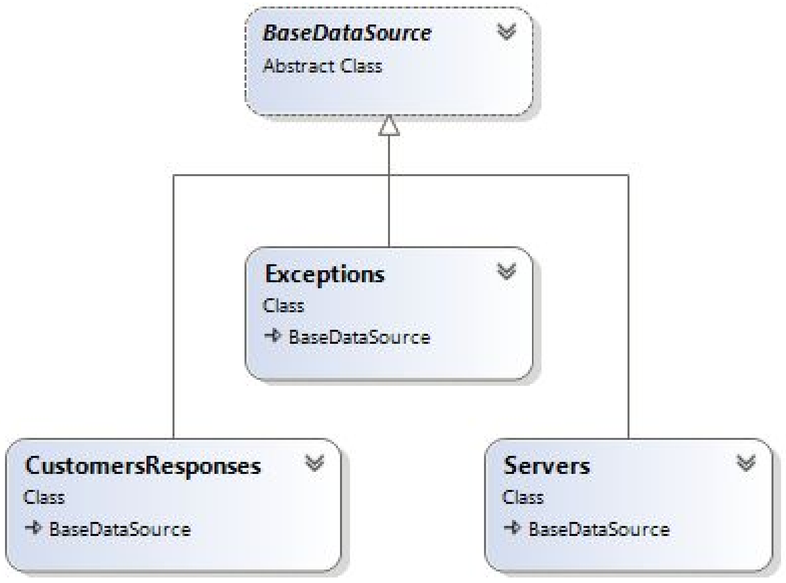


Figure 11. BaseDataSource diagram

BaseDataSource class consists of 3 child classes at the moment. They are used for each individual View. So if later it is needed to add a View that will contain other information, then another DataSource child class needs to be created. DataSources are used to get the needed data from the Third Party Resources and then transform the data into the needed format. Each data is individually gathered. If a DataSource does not need to include a particular Diagram, then the code needs to state it. Here is an example:

```
public override Diagrams.Map Map(List<BaseFilter> filters)
        {
            throw new NotImplementedException();
        }
```

Figure 12. DataSource unneeded diagram exception

This code is used in the Exceptions class to state that there is no need for a Map Diagram and that it will not be implemented.

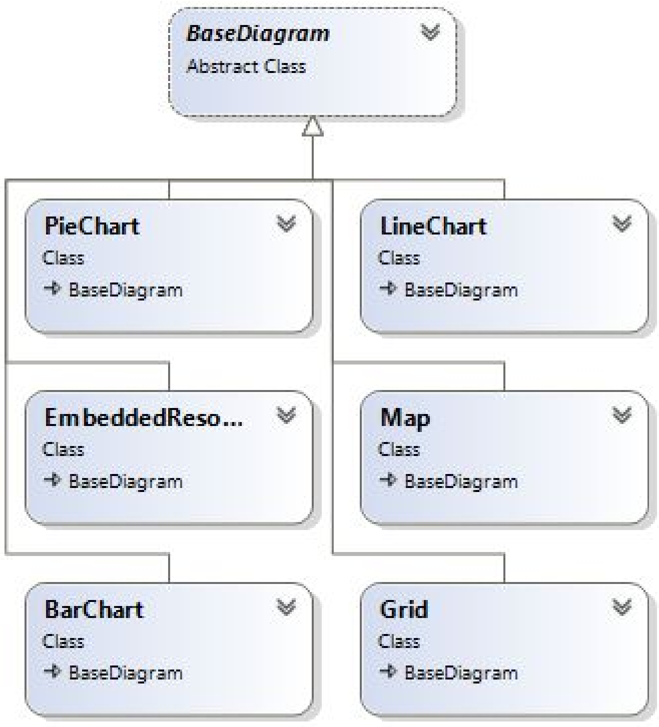


Figure 13. BaseDiagram diagram

BaseDiagram class has 6 child classes. Each child class is a type of Diagram. BaseDiagram class contains information that are needed for each Diagram: Name of the Diagram, so it can be later used to distinguish which Diagram is in use and Filters. Filters parameter is a list of BaseFilter class. Filters are used to filter information in the Diagram.

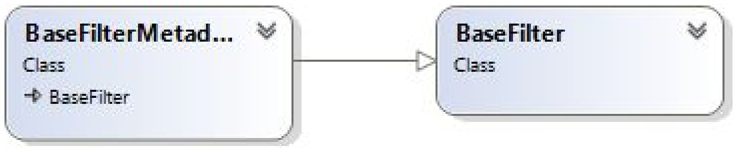


Figure 14. BaseFilter diagram

```
public class BaseFilter
    {
        public string Name;
        public string SelectedValue;
    }
```

Figure 15. BaseFilter class

BaseFilter carries information on the name of the Filter and the Selected Value for this Filter. However, additional information is needed, so the filters could work properly and it was decided to create an additional metadata class for filters.

```
class BaseFilterMetadata : BaseFilter
    {
        public List<string> PossibleValues;
        public Dictionary<string, string[]> DisabledValues;
        public string Dependant;
    }
```

Figure 16. BaseFilterMetadata class

Metadata contains PossibleValues, that are used to show other values that can be selected and then the diagram will be filtered by this value. Disabled Values and Dependant are used if a Diagram needs to use more than one filter and if they could contradict with each other. For example, in the Global View there is a Line Chart. It uses two Filters, one if which states the Time Frame and the other in which time units the Time Frame will be measured. By default Time Frame is set to “Last day”, so the user should not be able to choose a time unit more that is more than an hour. Dependant parameter is used to enter the name of the other Filter, that can contradict with the current filter.

4.3 Dashboard

When the client side is loaded, index.html file has a small script which calls the start() function from the dashboard.js file. Dashboard javascript file is a general script, which will build the Views and Visualisations. Start() function does the following:

```
self.start = function () {
            $.ajax({
                type: "GET",
                crossDomain: true,
                dataType: "json",
                url: opts.baseUrl + "/api/Dashboard/GetDashboards",
                data: null,
                success: function (dashboards) {
                    for (var i = dashboards.length - 1; i >= 0; i--) {
                        addDashboard(dashboards[i]);
                    }
                }
            });
        }
```

Figure 17. Dashboards initialization

This function will call a jQuery ajax GET request, which will send a request to the Controller. The data type is set json, so it is easier to work with in javascript. On success, this request will receive an array of different Views, which will be constructed one by one.

```
var addDashboard = function (metadata) {
            var $body = $('<div class="body-element body-element-' +
metadata.Id + '"></div>').prependTo(opts.$bodyContainer);
            $body.data("metadata", metadata);

            var $header = $('<div class="header-element header-element-'
+ metadata.Id + '"></span>').prependTo(opts.$headerContainer);
            $header.append('<button class="header-switch">' +
metadata.Name + '</button>');               $header.append('<button
class="header-lock"><i class="fa fa-unlock-alt"></i></button>');
            $header.append('<button class="header-refresh"><i class="fa
fa-refresh"></i></button>');
            $header.data("metadata", metadata);

            switchDashboard(metadata);

            for (var i = 0; i < metadata.Visualizations.length; i++) {
                var $visualization = $("<div class='visualization
visualization-" + metadata.Visualizations[i].Id + "'></div>");
                $visualization.data("metadata",
metadata.Visualizations[i]);

$visualization.width(metadata.Visualizations[i].Width+"%");

$visualization.height(metadata.Visualizations[i].Height+"%");

                var positionOpts = {
                    left: metadata.Visualizations[i].Left + "%",
                    top: metadata.Visualizations[i].Top + "%"
                };

                $visualization.css(positionOpts);
                $body.append($visualization);
            }
        }
```

Figure 18. Add Dashboard function

AddDashboard function will create a body container for the View, where all of the Visualisation of that particular View will be added. Each View container will have to add a button to the header container, so it would be easier to switch between them. Also, the lock button will be added. The lock button is used, so the user could stop the Views switching between each other and stay on one of them. After that, the refresh button is added. User needs to have an option to refresh the View manually and synchronise the data if needed. Views data is added as metadata to the body container, so it could be used later. To make this View active, the switchDashboard function is called. It will add an active class to the View, which means that it is currently displayed. This function is also called, when user tries to switch the View to another one or when the system automatically switches.

FOR cycle goes through each Visualisation in the View. The cycle creates a container for the Visualisation, adds metadata to the container, so it could be used later in the script and after that the dimensions are inserted, that were predefined by the developer at the backend, like the position options that are inserted later. After the Visualisation container is created, it is inserted to the Views body container.

When all of the Views and Visualisation containers are prepared, then the worker() function will be fired. It remembers the time it was fired and puts a timeout on the worker function, so it will be fired every N seconds. It is done for the dashboards to automatically switch between each other. When the switch is fired, then all of the Visualisation will be filled with charts and other diagrams. At the start, the View will not be switched, but instead only the Visualisation will be initialised.

```
var refreshVisualization = function (metadata, forceRefresh) {
            var $visualizationContainer =
opts.$container.find(".visualization-" + metadata.Id);
            if ($visualizationContainer.is(":visible") == true) {
                var isRefresh = forceRefresh;

                if (!isRefresh) {
                    if (metadata.Timeout <= 0) {
                        isRefresh =
$visualizationContainer.data("datasource") == undefined;
                    } else {
                        isRefresh = metadata.LastTimeout == null || new
Date().getTime() - metadata.LastTimeout.getTime() > metadata.Timeout
                    }
                }

                if (isRefresh) {
                    metadata.LastTimeout = new Date();

                    $.ajax({
                        type: "GET",
                        crossDomain: true,
                        dataType: "json",
                        url: opts.baseUrl +
"/api/Dashboard/GetDataSource",
                        data: {
                            dataSource: metadata.DataSource.Name,
                            diagram: metadata.Diagram.Name,
                            filters:
JSON.stringify(metadata.Diagram.Filters)
                        },
                        success: function (dataSource) {
                            $visualizationContainer.data("datasource",
dataSource);

                            var opts = { filters:
metadata.Diagram.Filters };


Diagrams[metadata.Diagram.Name].refresh($visualizationContainer,
metadata.Diagram, dataSource);

Diagrams[metadata.Diagram.Name].refreshHeader($visualizationContainer,
metadata.Diagram, dataSource);
                        }
                    });
                }
            }
        };
```

Figure 19. Refresh Visualisation function

Function refreshVisualization is called on each refresh or initial start. This function will continue only for those Visualisation that have a visible class. This means that the Visualisation will be built, if it is from a View that is currently active. ForceRefresh is used to identify if the function was called using the refresh button that the user pressed or by the system automatically. If the forceRefresh is set to true, it means that it was called by the user manually using the refresh button. If it passes the refresh check, then an ajax GET request is send to the API endpoint, which gets the data for that particular Visualisation. The GET request needs to send necessary data, so the backend could identify which data to send back. On success the Diagram will be built. Refresh function draws the Visualisation and the refreshHeader function draws the Filters for that Diagram if needed.

4.4 Visualizations

Each Visualisation is being constructed on a client side of the web application by the javascript libraries.

4.4.1 Grid

```
public class Grid : Base.BaseDiagram
    {
        public class Column
        {
            public string Name { get; set; }
            public string Type { get; set; }
            public int Width { get; set; }
            public string Align { get; set; }
        }
        public List<Column> Columns { get; set; }
        public object[] Rows { get; set; }

    }
```

Figure 20. Grid class

Grid class extends the Base Diagram class. Grid consists of 3 things: List of Columns information, Array of Row objects, Sort object for Sorting.

Column class has 4 fields: Name, Type, Width and Align. Name is used to determine which Grid the client side has received. Name is the name of the column, Type is used by the Grid plugin on the frontend to determine the type of the rows under the column. Width and Align are the dimensions that the developer can set for the particular column.

Rows is an array of objects, because the information in each object can be different depending on the number of columns and their names.

When the frontend downloads the dashboard information via an API endpoint, it will build the Views and necessary Visualisations. Code for the build of the Grid is as follows:

```
var Diagrams = Diagrams || {};

Diagrams.Grid = (function () {
    var diagram = function() {
        this.refresh = function ($container, diagram, dataSource) {
            if ($container.data('init')) {
                $container.jsGrid("insertItem",
dataSource.Rows[dataSource.Rows.length - 1]);
                $container.jsGrid("refresh");
            } else {
                var fieldsArray = [];

                for (var i = 0; i < dataSource.Columns.length; i++) {
                    if (!dataSource.Columns[i].CellStyle) {
                        fieldsArray[i] = {
                            name: dataSource.Columns[i].Name,
                            type: dataSource.Columns[i].Type,
                            width: dataSource.Columns[i].Width,
                            align: dataSource.Columns[i].Align
                        };
                    } else {
                        fieldsArray[i] = {
                            name: dataSource.Columns[i].Name,
                            type: dataSource.Columns[i].Type,
                            width: dataSource.Columns[i].Width,
                            align: dataSource.Columns[i].Align
                        };
                    }
                }

                $container.jsGrid({
                    height: "100%",
                    width: "50%",

                    sorting: false,
                    paging: false,

                    data: dataSource.Rows,
                    onItemInserting: function (args) {
                        args.grid.data.unshift(args.item);
                    },
                    onItemInserted: function (args) {
                        args.grid.data.splice(args.grid.data.length-1,
1);

                    },
                    fields: fieldsArray
                });

                $container.data('init', true);
            }
        }

        this.resize = function ($container, diagram, dataSource) {
            this.refresh($container, diagram, dataSource);
        }
    }

    diagram.prototype = new Diagrams.BaseDiagram();

    return new diagram();
})();
```

Figure 21. Grid initialization

On refresh, which is also called when the Visualisation needs to be drawn at the start, if clause will check if the Grid is already drawn, if yes, then it will simply insert the last row to the Grid. It was done like this, because if it was rebuilded, then the user, that was scrolling through the Grid, would lose the position, because the Grid will jump back to the start.
When the Grid should be drawn anew, then the code will assemble the column information properly for the Grid plugin to use. The Rows are already in the needed format, so it can be inserted as it is. The function that will draw the Grid can take few additional options like height, width, which are used to fill the Grid to its container. Sorting and Paging are set to false, because it was decided to not allow the Grid to be sorted, and the information should overflowed by a scroll and not by paging. There are also a few events used: onItemInserting and onItemInserted. Those events are used so later, if there is a need to add an additional row to the Grid on refresh, then the new row will be added to the top of all rows. Resize function is being called after window resize.

4.4.2 Pie Chart

```
public class PieChart : Base.BaseDiagram
    {
        public class DataSet
        {
            public string Label { get; set; }
            public double Value { get; set; }
            public string FillColor { get; set; }
        }

        public List<DataSet> DataSets { get; set; }
        public string Title { get; set; }
    }
```

Figure 22. PieChart class

Pie chart extends the Base Diagram class. PieChart class consists of only two parameters - List of DataSets and Title. Title is the name of the chart, in other words what data is displayed, so the user could know what he is looking at. DataSet is a class that represents one Pie Chart slice and consists of Label, Value and FillColor. Label is the name of the slice and Value is a numerical Value of the slice. FillColor should be manually selected by the developer, it will be the color of the Pie Chart Slice. Developer can choose a default colour from the Helper class, it will be described later, or enter his own. Once the front end gets the response from the API endpoint, it will generate the Views and later the Visualisations in them, Pie Chart initialization code is as follows:

```
var Diagrams = Diagrams || {};

Diagrams.PieChart = (function () {
    var diagram = function() {
        this.refresh = function ($container, diagram, dataSource) {
            var data = new google.visualization.DataTable();
            data.addColumn('string', 'Label');
            data.addColumn('number', 'Value');
            var colors = [];
            for (var i = 0; i < dataSource.DataSets.length; i++) {
                data.addRow([dataSource.DataSets[i].Label, 
dataSource.DataSets[i].Value]);
                colors.push(dataSource.DataSets[i].FillColor);
            }

            var options = {
                'title': dataSource.Title,
                'width': '100%',
                'height': '100%',
                'colors': colors,
                titlePosition: 'in',
                legend: {
                    position: 'bottom'
                }
            };

            var chart = new 
google.visualization.PieChart($container[0]);
            chart.draw(data, options);
        },

        this.resize = function ($container, diagram, dataSource) {
            this.refresh($container, diagram, dataSource);
        }
    }

    diagram.prototype = new Diagrams.BaseDiagram();

    return new diagram();
})();
```

Figure 23. PieChart initialization

Pie Chart uses the Google Charts JavaScript library, because of this the information needs to be transformed to a DataTable, before it can be drawn. Script creates two mandatory columns that are later filled in the FOR cycle. Each chart can have options that will specify needed custom modifications. Title will be displayed beside the chart, so users could know what kind of information is displayed. Width and Height are 100% by default, so the chart fully takes up the div container. Colors are being collected to an array, which later is being inserted to options object. Title position means where the title should be displayed, inside the chart or outside, in this instance it is inside the container. After all of this the chart is being drawn. Unlike the Grid, the Pie Chart is being redrawn on each refresh, it can be done, because redrawing the chart fully takes the same amount of time if the script would replace the old data with the new one. Resize function is called only when the window size is changed.

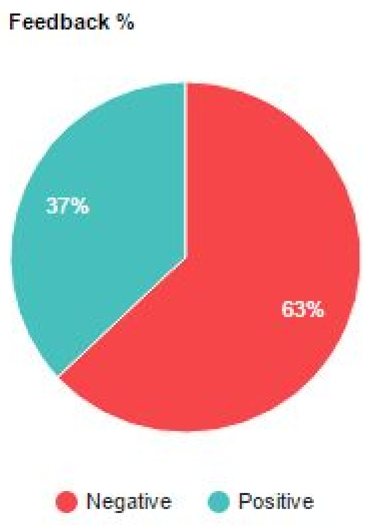


Figure 24. Pie Chart Example

4.4.3 Bar Chart

```
public class BarChart : Base.BaseDiagram
    {
        public class DataSet
        {
            public string Name { get; set; }
            public string FillColor { get; set; }
            public List<double> Data { get; set; }
        }

        public List<string> Labels { get; set; }
        public List<DataSet> DataSets { get; set; }
        public string Title { get; set; }
    }
```

Figure 25. Bar Chart class

Bar Chart class extends the Base Diagram Class. It has 3 parameters: Labels, DataSets and Title. Labels are a list of strings and consists of names of each Bar in the chart successively. Title is the name of the chart, so it can be distinguished from other charts in the View and so the user could know what information is displayed. DataSets are a list of a DataSet class. They represent each Bar data successively, meaning, that one DataSet contains data for one type of Bar. FillColor is the color of the Bar. The color for it can be selected from a Helper class, which will be described later, or a developer can manually enter the color code. Each Label can have any quantity of Bars, for example in the Generic View, each Label in the Bar Chart has 2 Bars: Positive and Negative. Data in DataSet contains the numerical values of all Labels in order, meaning that Labels and Data share the same index. Once the data is downloaded to the client side, it will run a script, that will draw the Bar Chart.

```
var Diagrams = Diagrams || {};

Diagrams.BarChart = (function () {
    var diagram = function () {
        this.refresh = function ($container, diagram, dataSource) {
            var arr = [];
            var labelArray = ["Label"];
            var colors = [];
            for (var i = 0; i < dataSource.Labels.length; i++) {
                var subArray = [];

                subArray.push(dataSource.Labels[i]);

                for (var j = 0; j < dataSource.DataSets.length; j++) {
                    if ($.inArray(labelArray,
dataSource.DataSets[j].Name) < 0 && arr.length == 0) {
                        labelArray.push(dataSource.DataSets[j].Name);
                        colors.push(dataSource.DataSets[j].FillColor);
                    }
                    subArray.push(dataSource.DataSets[j].Data[i]);
                }
                if (arr.length == 0) {
                    arr.push(labelArray);
                }
                arr.push(subArray);
            }

            var data = google.visualization.arrayToDataTable(arr);

            var options = {
                legend:{
                    position: 'bottom'
                },
                title: dataSource.Title,
                titlePosition: 'in',
                colors: colors,
                'width': '100%',
                'height': '100%'
            };

            var chart = new
google.visualization.BarChart($container[0]);
            chart.draw(data, options);
        },

        this.resize = function ($container, diagram, dataSource) {
            this.refresh($container, diagram, dataSource);
        }
    }

    diagram.prototype = new Diagrams.BaseDiagram();

    return new diagram();
})();
```

Figure 26. Bar Chart initialization

First the script needs to transform the received data to a multiarray that needs to represent a table, meaning that the first subarray needs to represent column name and subarrays after it will be rows. Later this array will be transformed to a DataTable. Script works this way, because Google Charts library draws the charts using the DataTable. Like the other charts, it will have options that customise the chart. Legend options states where the legends information should be displayed relatively to the chart. Title is the name of the chart. Title position will draw the title inside the chart, it is done, so it would not mess with the Filters, that will be added. Colors are the bar colors, they will not be individual for each Bar, only for each Bar Type. Width and Height are 100%, so the chart could fill the container. Resize only triggers after the window has been resized.

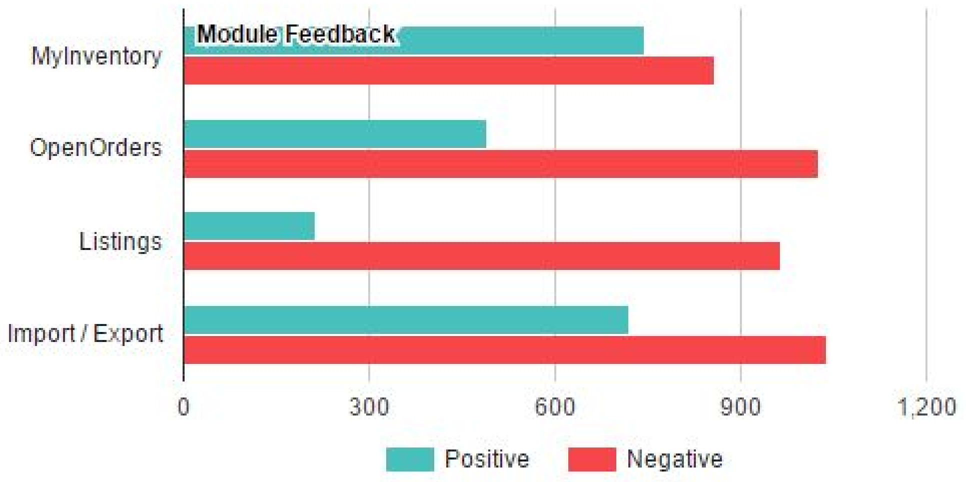


Figure 27. Bar Chart Example

4.4.4 Line Chart

```
public class LineChart : Base.BaseDiagram
    {
        public class Line
        {
            public string Name { get; set; }
            public Info Info { get; set; }
        }
        public class Info
        {
            public List<int> Data { get; set; }
            public List<string> Time { get; set; }
        }
        public string XName { get; set; }
        public List<Line> Lines { get; set; }
        public string Title { get; set; }
    }
```

Figure 28. LineChart class

Line Chart class extends the Base Diagram class. Line Chart consists of 3 parameters: XName, Lines and Title. XName will hold the name of the X axis of the Line Chart, it was decided not to hold the name of the Y axis, because the Title can have a name that will clear up what is displayed on the chart. If needed, it can be added later. Lines is a list of List objects that contain information on the Name of the Line and the necessary Info. Info class, which is in the Line class, will hold the numerical Data and their Time. Data and Time will have the same index, meaning that data on the first position will correspond to each other. To draw or redraw the chart, the following script needs to executed:

```
var Diagrams = Diagrams || {};

Diagrams.LineChart = (function () {
    var diagram = function() {
        this.refresh = function ($container, diagram, dataSource) {
            var data = new google.visualization.DataTable();

            data.addColumn('datetime', dataSource.XName);
            for (var i = 0; i < dataSource.Lines.length; i++) {
                data.addColumn('number', dataSource.Lines[i].Name)
            }
            var fullData = [];
            var timeData = [];
            for (var i = 0; i < dataSource.Lines[0].Info.Time.length;
i++) {
                var dataRow = [new
Date(dataSource.Lines[0].Info.Time[i])];
                timeData.push(new
Date(dataSource.Lines[0].Info.Time[i]));
                for (var y = 0; y < dataSource.Lines.length; y++){
                    dataRow.push(dataSource.Lines[y].Info.Data[i]);
                }
                fullData.push(dataRow);
            }
            data.addRows(fullData);
            var options = {
                title: dataSource.Title,
                hAxis: {
                    ticks: timeData
                },
                width:"100%",

                height: "100%",
                legend: {
                    position: 'top'
                },
                vAxis: {
                    gridlines: {
                        color: 'transparent'
                    }
                },
                hAxis: {
                    gridlines: {
                        color: 'transparent'
                    }
                }

            };

            var chart = new
google.visualization.LineChart($container[0]);
            chart.draw(data, options);
        },

        this.resize = function ($container, diagram, dataSource) {
            this.refresh($container, diagram, dataSource);
        }
    }

    diagram.prototype = new Diagrams.BaseDiagram();

    return new diagram();
})();
```

Figure 29. LineChart initialization

Before drawing the chart, the script needs to transform the needed information to a DataTable, so it can be later used to draw the chart. First of all, the script creates the column, where the first column will be X axis name and the others columns will be the Line Names, we also need to specify which type of data will be under the columns. After that, the rows will be added, while keeping in mind under which column the information will be inserted. Before drawing the Chart, some custom options will need to be specified. Line Chart has to be less cluttered, because of this, options takes two properties called vAxis and hAxis, where the grid line colors are transparent. It means that there will be no grid lines at the background of the chart and the only lines that will be visible are the X and Y axis and the data lines. Also, under hAxis, there is a property called ticks, it shows the positions on the chart, so the time or number could be correctly displayed on the X axis. Title is the name of the chart. Width and Height are 100%, so the chart could be filled into the container.

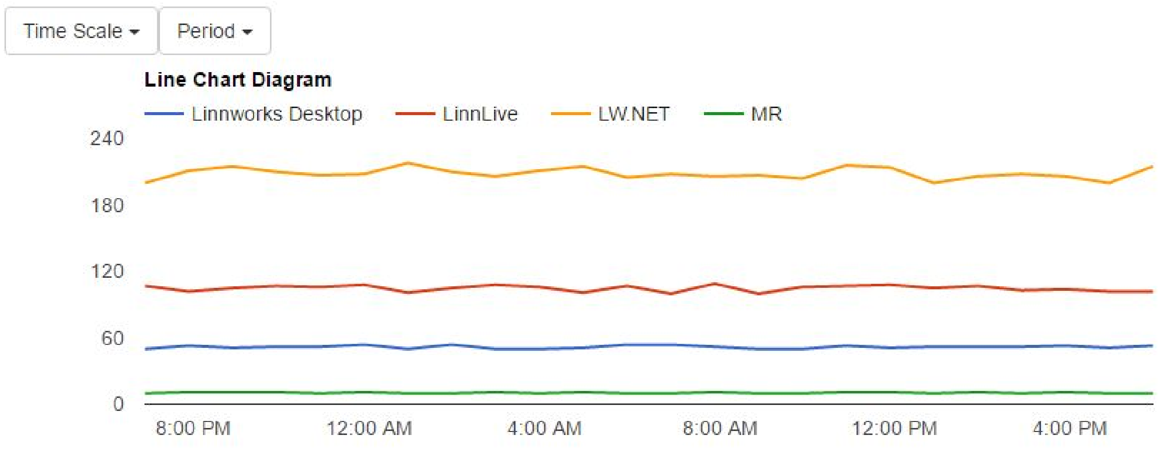


Figure 30. Line Chart Example with Filters

4.4.5 Map

```
public class Map : Base.BaseDiagram
    {
        public class Location
        {
            public string Country { get; set; }
            public int Popularity { get; set; }
        }

        public List<Location> Locations { get; set; }
    }
}
```

Figure 31. Map class

Map class extends the Base Diagram class. It contains only one parameter - Locations. Locations is a list of Location class. Location class contains information about each country. For example, the map on the Global View will hold only those countries information that have clients, which use our products. After downloading the required information, the following script will draw the Map:

```
var Diagrams = Diagrams || {};

Diagrams.Map = (function () {
    var diagram = function () {
        this.refresh = function ($container, diagram, dataSource) {
            var data = new google.visualization.DataTable();

            data.addColumn('string', 'Country');
            data.addColumn('number', 'Popularity');

            for (var i = 0; i < dataSource.Locations.length; i++) {
                data.addRow([dataSource.Locations[i].Country,
dataSource.Locations[i].Popularity]);
            }

            var options = {
                dataMode: 'regions'
            };

            var chart = new google.visualization.GeoMap($container[0]);
            chart.draw(data, options);
        },

        this.resize = function ($container, diagram, dataSource) {
            this.refresh($container, diagram, dataSource);
        }

    }
    diagram.prototype = new Diagrams.BaseDiagram();

    return new diagram();
})();
```

Figure 32. Map initialization.

On refresh script needs to transform the data into a DataTable, so the Google charts library could transform this information into a map. Script creates two columns which are called Country and Popularity, after that the DataTable is populated by rows. Options here are quite basic. There are no Width and Height, because the Map already tries to fit into the container. Unfortunately, there is one setback, there is no way to add title to the Map at the moment, because Google charts does not have an option parameter for that. DataMode means how the popularity will be displayed on the map. There are currently two options: Regions or

Markers. Regions means that the color of the country will depend on its popularity, it will become darker with more popularity. Markers means that on top of each country, that were specified in the received data, will be placed a marker, which will change its size and color, depending on the country's popularity. Author has chosen to use Regions, because it's more visually pleasing and reduces clutter, if the map will be used on a small portion of the page. After options declaration, the map is being drawn into the container. Resize function is triggered on windows resize.

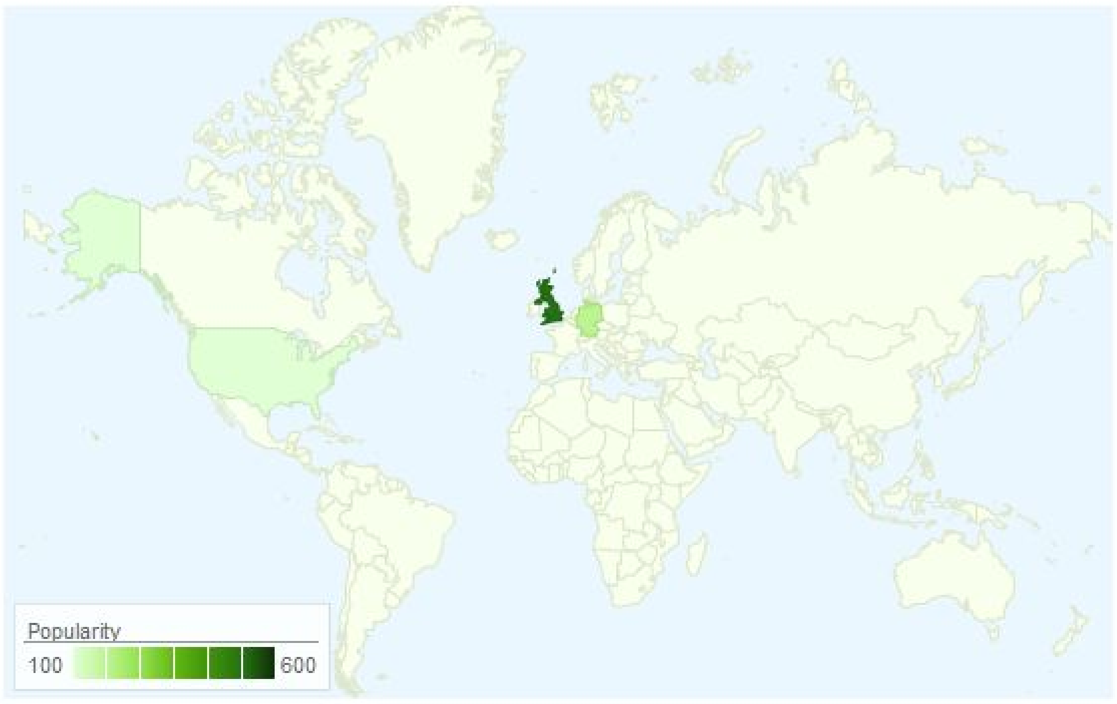


Figure 33. Map example. Countries used: USA, UK and Germany.

4.4.6 Embedded Resource

This diagram is used to insert diagram that are already drawn from third party softwares like Kibana. The class is constructed quite easy:

```
public class EmbeddedResource : Base.BaseDiagram
    {
        public string Content { get; set; }
    }
```

Figure 34. Embedded Resource class

It contains one parameter - Content. This third party resources will be displayed on the View using the HTML iframe tag. Iframe is used to embed another document within the current HTML document. Because of this, the Content parameter will contain an iframe tag with another HTML document. This particular Visualisation was developed, so the dashboard could show information from Kibana and Monitis.

4.5 Helpers

Helpers are used in LinnDashboard.Logic. They are there to facilitate the development of the web application. If a developer uses an icon, specific color or some function, that could be used somewhere else in the future, it is better to add them to Helpers. There are currently 3 files under Helpers folder: Helper class, Icons class and StandardColors class. Helper class consists of two functions at the moment.

```
public static string TruncateLongWords(string text, int length)
        {
            var splitText = text.Split(' ');
            var truncated = splitText.Select(s => s.Length > length ?
s.Substring(0,length)+"..." : s);
            return string.Join(" ", truncated);
        }
```

Figure 35. Truncate long words function

This functions is used at the moment for the Grid. There was a problem, while developing the Grid, that if the word is too long, then it would mess with the width and height of the Grid inside the container. This function fixed this problem, by truncating long words to the specified length.

```
public static string NewLine(string text)
        {
            return text.Replace(System.Environment.NewLine, "<br>");
        }
```

Figure 36. Functions that creates a new line in a string

This is a quite easy function, which replaces new lines with html tag <br>. It was also implemented for the grid, because it did not recognise the new lines that were in the received text.

Icons class is used for frequently used Font Awesome icons. Font Awesome is a css library that gives developers scalable vector icons. It is done so the web application would have some consistency and there would not be, for example, 3-4 different icons for one thing. Currently there are 3 icons that can be returned. Here is an example of one:

```
public static string Positive
        {
            get
            {
                return "<i class='fa fa-smile-o response-icon' 
style='color:"+Helpers.StandardColors.Green+"'></i>";
            }
        }
```

Figure 37. Positive icon static string

This function returns a ‘Positive’ icon, which is colored green.

StandardColors class is used for the same reason as the Icons, so there would not be many different shades of the same color, then it would bring consistency to the UI. Example of one function:

```
public static string Red
        {
            get
            {
                return "#F7464A";
            }
        }
```

Figure 38. Red color static string

This function returns a hex code for the red color.

4.6 DataSource

DataSource is a class that consists of functions that gather data from Third Party resources. Each View will have its own class that will gather info for each Visualisation. Each DataSource will have information about each Diagram, if there is no need for a particular diagram, then it has to be stated in the code by throwing an exception:

```
public override Diagrams.Map Map(List<BaseFilter> filters)
        {
            throw new NotImplementedException();
        }
```

Figure 39. Unneeded Diagram example

This function states that this particular DataSource does not need to use a Map diagram.

If a specific diagram is needed, then the DataSource for the View will gather information from a Third Party resource, transform the data to the diagram object and return this object. Example of an Embedded Resource diagram:

```
public override Diagrams.EmbeddedResource
EmbeddedResource(List<BaseFilter> filters)
        {
            var resource = new Diagrams.EmbeddedResource();

            resource.Content =
            @"
            <iframe
src=""http://logstore.linnworks.net/app/kibana#/dashboard/Exceptions?emb
ed&_g=(refreshInterval:(display:'1%20minute',pause:!f,section:2,value:60
000),time:(from:now-24h,mode:quick,to:now))&_a=(filters:!(),options:(dar
kTheme:!t),panels:!((col:1,columns:!(message,_type,database,module),id:E
xceptions,row:5,size_x:12,size_y:5,sort:!('@timestamp',desc),type:search
),(col:1,id:%5BLinechart%5D-Exceptions-count-per-type-and-per-time,row:1
,size_x:7,size_y:4,type:visualization),(col:8,id:%5BVertical-Bar-Chart%5
D-Exceptions-per-type-and-per-module,row:1,size_x:5,size_y:4,type:visual
ization)),query:(query_string:(analyze_wildcard:!t,query:'*')),title:Exc
eptions)""
                height=""100%""
                width=""100%"">
            </iframe>
            ";

            return resource;
        }
```

Figure 40. Embedded Resource Data Source example

This DataSource function returns a Embedded Resource object, which contains an iframe tag for the company Kibana Dashboard.

4.7 Controllers

Controllers are used in the LinnDashboard.API project to catch request and return responses. There are only two API endpoints that are used, because there is no need to use more than that.

One request returns the Dashboards that contain all the information for each Visualisation in the View.

```
[ActionName("GetDashboards")]
        public List<LinnDashboard.Logic.Base.BaseDashboard>
GetDashboards()
        {
            return LinnDashboard.Logic.Adapter.GetDashboards();
        }
```

Figure 41. Get Dashboards API endpoint

This function calls another function in the Adapter class, which gathers information for all dashboards.

After a View is built on the client side, then request GetDataSource will be sent from each Visualisation from that View. GetDataSource will return data for the diagram.

```
[ActionName("GetDataSource")]
        public LinnDashboard.Logic.Base.BaseDiagram GetDataSource(string
dataSource, string diagram, List<BaseFilter> filters)
        {
            return LinnDashboard.Logic.Adapter.GetDataSource(dataSource,
diagram, filters);
        }
```

Figure 42. Get Data Source API endpoint

This function calls a function in the Adapter class and passes parameters 3 parameters, so the function in the Adapter could determine from which DataSource class the diagram originated, the type of Diagram and what filters are currently active.
After all of the information is gathered, the server will send the response back.

4.8 Adapter

Adapter class is used to prepare the Dashboards information and the gathered data from the Data Sources. There are two functions in the Adapter class: GetDashboards and GetDataSource.
GetDashboards is called after the controller receives a request from the client side.

```
var dashboards = new List<Base.BaseDashboard>();
var global = new Base.BaseDashboard();
global.Name = "Global";
global.Visualizations = new List<Base.BaseVisualization>();

global.Visualizations.Add(new Base.BaseVisualization()
            {
                DataSource = new DataSource.CustomersResponses(),
                Diagram = new Diagrams.LineChart()
                {
                    Filters = new List<Base.BaseFilter>
                    {
                        new Base.BaseFilterMetadata()
                        {
                            Name = "Period",
                            SelectedValue = "Last day",
                            PossibleValues = new List<string>()
                            {
                                "Last day",
                                "Last week",
                                "Last month",
                                "Last year"
                            }
                        },
                        new Base.BaseFilterMetadata()
                        {
                            Name = "Time Scale",
                            SelectedValue = "Hour",
                            PossibleValues = new List<string>() {
                                "Hour",
                                "Day",
                                "Month"
                            },
                            DisabledValues = new Dictionary<string,
string[]>
                            {
                                {"Last day", new string[] {"Day","Month"
} },
                                {"Last week", new string[] {"Month" } },
                                {"Last month", new string[] {"Month" } }
                            },
                            Dependant = "Period"
                        }
                    }
                },
                Width = 50,
                Height = 30,
                Timeout = 10000,
                Left = 0,
                Top = 35
            });

dashboards.Add(global);

return dashboards;
```

Figure 43. Get Dashboards function example

Function creates a list of dashboards that will be later returned. Dashboard object is created and given a name. This name will be shown on the client side. After that the list of Visualisation is created. Each Visualisation will be placed there, because the way the Visualisations are generated is almost the same for each Visualisation. Author decided to show only one example, in this instance it is the Line Chart diagram. BaseVisualisation needs a few parameters that will state from which DataSource the diagram is, what type of Diagram it is, the position of the Visualisation inside the View, dimensions and the time frame of when the Visualisation will be refreshed. Filters parameter is added to the Diagram and at start will be the Default filters for it. If the user select a different value for the filter on the client side, then only the Select Value will be changed. After the Visualisations are ready and are inserted into the dashboard object, it will be added to the list of dashboards.

Other function that the Adapter class has is GetDataSource. This function is used on each Visualisation initialisation or refresh, when a request is sent from the client side. It will get the data for a particular diagram from its DataSource class.

```
public static Base.BaseDiagram GetDataSource(string dataSource, string
diagram, List<Base.BaseFilter> filters)
        {
            var dataSourceType =
Type.GetType("LinnDashboard.Logic.DataSource." + dataSource);
            if (dataSourceType == null)
                throw new Exception("DataSource doesn't exist.");

            var dataSourceInstance =
Activator.CreateInstance(dataSourceType);

            var parameters = new object[1] { filters };

            var diagramCall = dataSourceType.GetMethod(diagram);
            var diagramCallResult =
diagramCall.Invoke(dataSourceInstance, parameters) as Base.BaseDiagram;

            return diagramCallResult;
        }
```

Figure 44. Get Data Source function

This functions determines from which DataSource the Visualisation needs to get its data. If the data source if found, then an instance will be created. The filters will be passed as parameters to the diagram function. The diagram function, that is located in the specific Data Source class, needs to be found, so it could be called later and the result will transformed to the BaseDiagram object. After the data for that Visualisation is ready, it will be returned to the Controller and then sent to the client side as a response.

4.9 Future development

Author of this project will keep developing it in the future, because the company has plans on how to make this web application better, so it could be more beneficial to the department's workflow and teamwork.

The plan is to make a separate View for each department, where information about the workflow and achievements of this department will be displayed. Team Leaders of each department will come up with what to show on their Views and will forward this information to the author of this project. Each department will be able to update static Visualisations, for example news, without prior knowledge of coding.

The reasoning behind this plan is to inform staff of other departments on what will be and has been done by the department. This will help in teamwork building, because every staff member will know how the work of each department is connected. The problem of many companies is that people don’t consider that the work that they are doing is important for other departments workflow. Implementation of this Views should help with fixing this issue.

After implementing Views for each department, there is another feature that needs to be added. The feature is aimed to motivate people, by adding some kind of achievement system to each department View. While researching on how to motivate people, many sources state that one of the best systems of motivation is gamification. [23][24] Achievements need to relate with the work the people are doing and should relate to the department. Examples are, giving an achievement to a developer who pushed over N lines of new code to the Mercurial branching system or giving an achievement to a support staff, who closed N number of tickets in a week. This should keep people motivated. This is still being looked at and will be fully agreed on after the first plan will be ready to use.

5. Summary

This version of the Linnworks Dashboard is not final, author will keep developing it further and try different approaches to try keeping various company departments informed. The current version of the dashboard will help companies staff to be updated on current feedback from clients, how often our products are being used and in which countries. Our staff will be able to get feedback to their work.

The development and the final version have achieved every functional and nonfunctional requirements. Throughout the development author experimented with different libraries and methods to develop the needed dashboard functionality. Project was build in a way that other developers could easily navigate through the code and add their modifications or Views/Visualisations with ease. Dashboard is currently used in our office and displayed on a Smart TV in each department room. If needed, our staff can also go to the site, where the dashboard is located and use it on their computer.

The project will continue to grow and the author will continue to develop it. There are plans on adding a separate View for each department, so our staff could keep up with not only their department work and also other departments work. With this, we hope to achieve understanding between each department, in what they are doing for the company. Other plan is to find a way to motivate staff by adding gamification to the dashboard in the form of achievements.

Because of this project our staff now know visually how many clients are using our software day by day and what they think of it. Naturally, the support department has knowledge on the feedback the clients provide, but other departments were not interacting with clients, therefore they did not know how massive the client base is for such a small company, that grows rapidly, but now they are informed and know that the work they do is important.

5. Kokkuvõte

Käesoleva töö raames loodud tarkvara Linnworks Dashboard ei ole selle lõplik versioon. Autor soovib edaspidi jätkata selle arendamist ja lõpuks panna kokku midagi uut selliseks, et kõik osakonnad oleksid ettevõttes toimuvaga kursis. Tänane versioon aitab töötajatel saada informatsiooni selle kohta missugune tagasiside tuleb klientidelt, kui tihti tooteid kasutatakse ja mis riikidest on selle kasutajad. Ettevõtte töötajad ja arendatava tarkvara kasutajad võivad saada rakenduse abil tagasisidet oma töö kohta.

Autor kasutab oma töös vajalikku funktsionaalsuse arendamiseks erinevaid raamatukogusid ja meetodeid. Projekt oli ehitatud selliseks, et ka teised ettevõtte arendajad saaksid koodi ümber kirjutada ja lisada oma muudatusi Views / Visualiseerimises. Rakendus on praegu kasutusel LinnSystems kontoris ja selle näidatakse Smart TV igas osakonna toas. Vajaduse korral, firma töötajatel on võimalik pääseda ka rakenduse asukohale ning kasutada selle oma arvutis.

Projekt on pidevalt kasvav ja lõputöö autori eesmärgis on selle edaspidine arendamine. Plaanides on lisada rakendusele erinevaid liideseid sellisteks, et ettevõtte personaal oleks võimeline saada informatsiooni mitte ainult oma osakonna tööst, aga ka teiste osakondade töö tulemusi. Plaanis on realiseerida töötajate motiveerimisvõimalusi lisades gamification'i.

Käesoleva lõputöö tulemuseks on arendatud tarkvara, mille abil ettevõtte töötajad saavad teada kui palju inimesed kasutavad käesoleva tarkvara igal päeval ja kuidas nad selle hindavad. Loomulikult tänapäeval tugiosakond on informeeritud sellest, missugune tagasiside tuleb klientidel poolt. Kuid siiamaani teised osakonnad, mis ei suhelnud klientidega otseselt, ei teadnud kui suur kliendibaas on firmal. Kliendibaas kasvab nii kiiresti. Autori poolt loodud tarkvara kasutamise tagajärjel kõik töötajad teavad nüüd et nende töö on ikka oluline.

References

[1] "Order management software," in *Linnworks*. [Online]. Available: http://www.linnworks.com/. Accessed: May 20, 2016.

[2] "Kibana," in *Elastic*, 2016. [Online]. Available: https://www.elastic.co/products/kibana. Accessed: May 20, 2016.

[3] "Network & IT systems monitoring – Monitis," in *Monitis*, 2006. [Online]. Available: http://www.monitis.com/. Accessed: May 20, 2016.

[4] "Monitis API documentation," in *Monitis*. [Online]. Available: http://www.monitis.com/docs/api.html. Accessed: May 20, 2016.

[5] "Logstash," in *Logstash*, 2016. [Online]. Available: https://www.elastic.co/products/logstash. Accessed: May 20, 2016.

[6] "TypeScript," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/TypeScript. Accessed: May 20, 2016.

[7] G. A. Wix, "Developing large-scale applications with typeScript," in *JavaScript*, Wix Engineering, 2015. [Online]. Available: http://blog.wix.engineering/2015/09/30/developing-large-scale-applications-with-typescript/. Accessed: May 20, 2016.

[8] M. Wasson, "Getting started with web API 2 (C#)," The Official Microsoft ASP.NET Site, 2015. [Online]. Available: http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api. Accessed: May 20, 2016.

[9] "JQuery," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/JQuery. Accessed: May 20, 2016.

[10] jq. Foundation, "JQuery's Ajax-Related methods," 2016. [Online]. Available: https://learn.jquery.com/ajax/jquery-ajax-methods/. Accessed: May 20, 2016.

[11] "AngularJS," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/AngularJS. Accessed: May 20, 2016.

[12] "What does AngularJS do better than jQuery?," 2016. [Online]. Available: http://stackoverflow.com/questions/18414012/what-does-angularjs-do-better-than-jquery. Accessed: May 20, 2016.

[13] "Chart.js,". [Online]. Available: http://www.chartjs.org/. Accessed: May 20, 2016.

[14] D. Phipps, "Interactive JavaScript charts for your webpage," 2016. [Online]. Available: http://www.highcharts.com/. Accessed: May 20, 2016.

[15] "Charts | Google developers," Google Developers. [Online]. Available: https://developers.google.com/chart/. Accessed: May 20, 2016.

[16] mleibman, "Mleibman/SlickGrid," GitHub, 2016. [Online]. Available: https://github.com/mleibman/SlickGrid. Accessed: May 20, 2016.

[17] A. Tabalin, "Lightweight grid jQuery Plugin," - jsGrid, 2015. [Online]. Available: http://js-grid.com/. Accessed: May 20, 2016.

[18] "Mercurial," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Mercurial. Accessed: May 20, 2016.

[19] S. Borho, "TortoiseHg," 2016. [Online]. Available: http://tortoisehg.bitbucket.org/. Accessed: May 20, 2016.

[20] Atlassian, "Free mercurial and git client for windows and Mac," 2016. [Online]. Available: https://www.sourcetreeapp.com/. Accessed: May 20, 2016.

[21] "Mobile App development & App creation software,". [Online]. Available: https://www.xamarin.com/. Accessed: May 20, 2016.

[22] "Project rider – A C# IDE," 2016. [Online]. Available: https://blog.jetbrains.com/dotnet/2016/01/13/project-rider-a-csharp-ide/. Accessed: May 20, 2016.

[23] T. Klosowski, "The psychology of Gamification: Can Apps keep you motivated?," Lifehacker, 2014. [Online]. Available: http://lifehacker.com/the-psychology-of-gamification-can-apps-keep-you-motiv-1521754385. Accessed: May 20, 2016.

[24] R. Stanley, "Top 25 best examples of Gamification in business | ClickSoftware Blogs," ClickSoftware Blogs, 2014. [Online]. Available: http://blogs.clicksoftware.com/index/top-25-best-examples-of-gamification-in-business/.

Accessed: May 20, 2016.