DELFT UNIVERSITY OF TECHNOLOGY

COMPUTER SCIENCE
TI3806 BACHELOR END PROJECT

Project Delphi

[redacted]

What is innovation?

Lex Boleij
Weilun Chen
Toine Hartman
Marciano Jorden

supervised by
Odette SCHARENBORG (TU Delft; Coach)
Otto VISSER (TU Delft; BEP Coordinator)
Huijuan WANG (TU Delft; BEP Coordinator)
[redacted] (*Company X*; client)
[redacted] (*Company X*; client)
[redacted] (*Company X*; client)

8th July 2019

Preface

With respect to confidential information at the company, the company will me referenced to as 'Company X', and all information that could lead back to the client are blacklined.

This document describes the Bachelor Thesis created by Lex Boleij, Weilun Chen, Toine Hartman and Marciano Jorden as a part of the Bachelor Computer Science at Delft University of Technology. During eleven weeks, the team created a web application for *Company X* that is able to visualise data that is scattered over multiple platforms, in a dynamic, functional and interactive fashion.

We would like to take this opportunity to thank Odette Scharenborg for her enthusiastic coaching and support for the duration of the project as well as the feedback she has given us on the different reports made.

Also, we want to thank [redacted], [redacted] and [redacted] for their creativeness and time to make this project as successful as it was.

L. Boleij
W. Chen
T.J.B. Hartman
M.R.F. Jorden
Delft, June 2019

Abstract

Being a company with over 1500 employees, a lot of data is available about people and their day-to-day pursuits, including projects they are working on. *Company X* has requested to gain more insight into this data, as it is currently scattered over multiple systems. Specifically, they requested to gain insight into which projects have been started around a certain topic, who is involved in these projects, what the status on these projects is and where these projects are carried out inside the company. *Company X* wants this data represented in a dynamic, scalable and interactive visualisation.

To create a product that fulfils the expectations and needs of *Company X*, a Proof of Concept (PoC) was created. This basic version was used to make sure that *Company X* and the development team had the same basic idea of the application to be created and was then extended to an Minimal Viable Product (MVP). Having completed this MVP, additional features were implemented. Such features include making the solution generic for different data sources, the ability to filter and search through the data and highlighting/focusing specific data types.

In order to ensure that the visualisation can be used instinctively and deliver the information it is meant to, a user study was set up. In this user study, multiple employees of *Company X* tested the visualisation by completing a set of tasks within the application. By tracking the different results (i.e. number of clicks, time, subjective opinions), the team was able to derive that the application is quite promising, but still had the need for some improvements to explore its full potential.

Executive Summary

As *Company X* has a lot of data about employees and their day-to-day business scattered over multiple systems, they proposed to create a visualisation of that data. The product described in this report, a web application for internal use by *Company X*, visualises the relations between (groups of) employees, innovative projects they work(ed) on, and other similarities between them. This enables employees at *Company X* to answer questions like 'Who works on innovation?', 'Where is this happening?', but also find implicit relations like which offices or teams of employees are often concerned with innovation or which employee would be suitable to conduct a specific project.

To make sure the development team and *Company X* were on the same page regarding functionalities that the product should have, a list of requirements was assembled according to the MoSCoW-principle, which can be found in section 2.3. Following these requirements, a Proof of Concept (PoC) was created, to find out whether the solution in mind would give the intended results. Next, this PoC was further developed into a Minimal Viable Product (MVP), where all of the must-haves would be implemented. The result of the MVP was a basic version of the product, showing a visualisation of projects and employees of *Company X*. However, as the access to real company data was delayed, this version used dummy-data. After this, the MVP was extended with additional functionalities, being a filter function, in order to reduce the amount of data shown in the visualisation, and a data import function, which is completely generic (data content agnostic) and thus makes as few assumptions about the data as possible without any complications. The implementation of this solution consists of three parts, being the back-end (subsection 4.1.2), the front-end (subsection 4.1.3) and a visualisation (subsection 4.1.4). The back-end is implemented in Ruby, with the Rails framework for database management and serving web pages. The *Vue.js* framework was used to develop the front-end of the application. This is a popular JavaScript (JS) framework that uses *Vue.js* Single File Components (SFCs), which organise the HTML (layout), CSS (styling) and JS (behaviour) belonging to a single *component* in a single file. A *Vue.js* component can be re-used multiple times and can anticipate writing large amounts of similar code. The visualisation is a major part of the front-end display that users will see and interact with. D3.js is a JS based library. Its main purpose is the manipulation of data into an interactive visualisation using HTML, Scalable Vector Graphics (SVG) and Cascaded Style Sheets (CSS). D3.js achieves this by allowing you to bind arbitrary data into the Document Object Model (DOM), and then transforms the object by data-driven transformation.

To make sure the application works as intended, a user study was performed, for which the question to answer was ***"To what extent can the visualisations, i.e. explicit and compact, be intuitively grasped and provide the information they are supposed to provide?"***

In order to investigate this question, the following sub-questions need to be answered.

- When is a visualisation successful?
- How does the user experience the current visualisation?
- How can the result of the user study be used to improve the visualisation?

After this user study, where seven people participated, the conclusion was made that the application succeeds to a certain extent, but quite some improvements need to be made in order to explore its full potential. As there was time left after the user study, some of the feedback that was given could already be implemented, while the rest of the improvements will be left for future implementation.

Although the product is useful as-is, improvements can be made (section 8.1: Recommendations); a real-time link with actual data reduces the need for human intervention in importing data, and the filter and 'compact' context require faster performance to be usable on larger datasets.

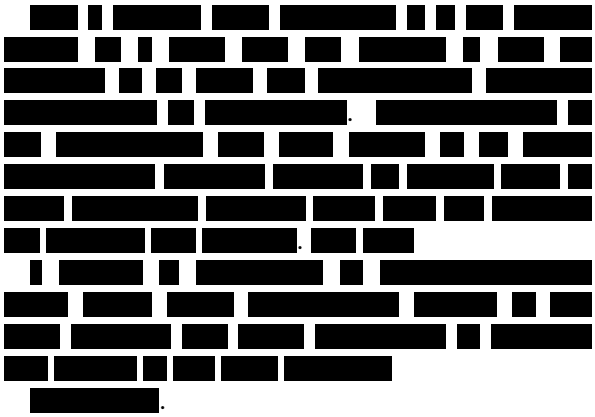
With regards to the original problem statement defined in section 2.1, the final product does bring the scattered data together in a dynamic, scalable and interactive fashion.

List of Figures

List of Tables

Contents

1 Introduction



Being a company with over 1500 employees, *Company X* has a lot of data about employees and their day-to-day pursuits including projects they are working on. They have requested to gain more insight into this data, as it is currently scattered over multiple systems (e.g. the Document Management System (DMS), Active Directory (AD), the Identity Management Solution and the SharePoint (SP)[1] based corporate intranet). This project should make it possible to visualise the relations between these data in a dynamic, scalable and interactive fashion. Specifically, they requested to gain insight into

1. which projects have been started around a certain topic,
2. who is involved in these projects,
3. what the status of these projects are, and
4. where these projects are carried out (e.g. which home market, which business unit).

The solution has to be accessible via web browser and will be used by all *Company X*'s employees. The team will also have to decide whether it is necessary to implement permission or role based governance for users.[2]

In order to ensure that the visualisation can be used instinctively and deliver the information it is meant to, a user study will be performed. In this study, multiple employees of *Company X* will test the product by completing a set of tasks within the application and going through multiple visualisations. By tracking the different results (i.e. number of clicks, time, subjective opinions), the team will try to answer the question:

"To what extent can the visualisations, i.e. explicit and compact, be intuitively grasped and provide the information they are supposed to provide?"

The explicit and compact visualisation will be described in subsection 4.1.1. In order to endorse this question, the following sub-questions need to be answered.

- When is a visualisation successful?
- How does the user experience the current visualisation?
- How can the result of the user study be used to improve the visualisation?

This report will first define and analyse the faced problem in chapter 2. The solution that the team provided will be explained in chapter 3, where the process and the design of the product will be elaborated, as well as how it solves the provided problem and what the impact is on *Company X*. Chapter 4 will dive further into the implementation of the product, split in the back-end and the front-end of the application and will describe how the team ensured the code was maintainable and well-documented. Chapter 5 will describe the research that was performed in order to assess to what extent the product design was intuitive and whether it provides the information it is supposed to. This report will come to an end with a conclusion (chapter 7), a discussion (chapter 8) on recommendations and ethical implications and finally a reflection (chapter 9). While some parts of the research report from week 2 have been integrated into this final report, the research report has been added as Appendix I for reference.

[1] *What is SharePoint?* URL: `https://support.office.com/en-ie/article/what-is-sharepoint-97b915e6-651b-43b2-827d-fb25777f446f`.

[2] Based on `https://bepsys.ewi.tudelft.nl/course_editions/7/projects/275` (visited on 24/05/2019)

2 | Problem

This chapter will define the problem that was proposed by *Company X* and will give an analysis of the problem including the requirements that were constructed.

2.1 Definition

As briefly mentioned in chapter 1, a lot of data is available at *Company X*, but there is currently no way of viewing this data in a clear way. The data contains records of employees and what their day-to-day pursuits are, including projects and personal and professional interests. The following data is available on the *Company X* intranet:

Employees. Basic employee information (*name*, *date of birth* etc.), work related information ([redacted], *room number*), personal interests, former projects.

Projects. A list of projects being worked on, containing information like *Project Owner*, *Project Members* and other metadata.

Practice Groups. A Practice Group (PG) is a department within *Company X*. Employees who belong to a PG work on a certain specialisation. Examples of practice groups are *Banking & Finance*, *Real Estate* and *Litigation & Risk Management*. An employee is associated with a single PG. PGs generally operate on all disciplines (legal, notarial and tax) within their scope.

Knowledge Groups. A Knowledge Group (KG) is a (sub)market of clients, e.g. *Healthcare* or *Automotive*, which are used internally at *Company X*. It contains employees from multiple PGs.

Profession groups. A Profession Group expresses in which discipline an employee works. *Company X* features the profession groups *Legal*, *Tax*, *Notarial* and *General* (also known as *Business Services*, i.e. *Finance*, *IT*, *HR* among others).

This data is available on the following platforms, which are some of the platforms that are most likely interesting for the product:

Inside is the intranet SP page, accessible by all *Company X* employees and houses company news, links to resources and the profiles of employees.

Knowhow. [redacted] is the SP page that contains Knowhow. Knowhow contains several lists, one of which is referred to within the company as '*The List*'. '*The List*' consists of projects that involve innovation within the company.

ICTweb is a SP that contains a list of projects carried out at *Company X*.

Employeedata (*Odata-feed*, *JSON*) is an OData[1] Application Programming Interface (API) that serves stored data of employees in JavaScript Object Notation (JSON) format.

The product has to bring these different platforms together in a visualisation.

2.2 Analysis

The goal of the product is that it should be possible to visualise the relations between the available data in a dynamic and interactive fashion in order to make the data more insightful. For example: When someone wants to determine the best fit for future projects, they want to use information about employees such as their professional interests and which projects they have previously worked on.

Furthermore, *Company X* has asked that the product be as generic as possible, as they want to import different data sources in the future.

The criteria to make this product a success are specified in a list of requirements in section 2.3.

2.3 Requirements

In order to build the right product for the described problem a set of requirements was compiled. These can either be classified as *functional* or *technical*.

Functional requirements specify needs that are visible to the end user, e.g. capabilities of the product or specifics on the layout. *Technical requirements* specify 'invisible' constraints, e.g. requirements on the development process or code quality.

[1] `https://www.odata.org/`

The requirements are prioritised conform the *MoSCoW*[2] method. This method is used to separate the requirements into *must-haves*, *should-haves*, *could-haves* and *won't-haves*.

Must-haves are requirements critical for developing the right product. Without them the product would not be right for the described problem. **Should-haves** are less important than *must-haves*, but still add a lot of value to the product. **Could-haves** have less importance than *should-haves* and have a lower value to development time ratio or have dependencies that are uncertain at this moment. **Won't-haves** will not be implemented, because they do not fit in the scope of this Bachelor End Project (BEP) or the defined problem.

2.3.1 Functional requirements

Must have

Link data from existing data sources. Without this link we can not have an up-to-date representation of the company data.

Overview of projects with statuses. The main purpose of the product is to aggregate and show information about existing projects and its employees.

Overview of people in projects with interests and expertises. The main purpose of the product is to aggregate and show information about existing projects and its employees.

Show projects that involve innovation. The company explained that they intend to use this product to show what the company is doing with regard to innovation.

Detect and alert about conflicting data from different sources. Aggregating data might yield duplicate records, and these records might not be identical if one of the data sources is not kept up-to-date (or if there are typos). By detecting conflicting data, steps can be made to make the shown data more accurate.

[2] D. Clegg and R. Barker. *CASE Method Fast-track: A RAD Approach*. CASE method. Addison-Wesley Publishing Company, 1994. ISBN: 9780201624328.

Accessible from the company's intranet. The final product is intended to be used by the companies' employees. The intranet shields from access from outside the company.

Accessible with *Azure Active Directory* Single Sign On (SSO). Company employees already actively use an *Azure* AD account, so this integrates well with the company's software infrastructure. Having to create a separate account would discourage employees from using the product.

User permissions. Not everyone should have access to all data. Some data is not meant to be seen by everyone.

Should have

A visual graph linking projects to users. To create a clear overview of the information, a visual layout is an effective and intuitive solution. An overview is possible in a static HyperText Markup Language (HTML) format, but lacks the interactive elements. Therefore, a hands-on interactive visualisation should be integrated. Depending on the element (e.g. a project or a user) that has the focus relevant information and links to neighbouring elements should be displayed. Irrelevant information should be hidden to keep the layout comprehensible.

Generic solution for adding new unspecified data sources. The company has requested that they might want to add new data sources in the future. As we cannot know the specific structure of these sources in advance a generic solution would be optimal. The solution should accept new data about or linked to projects and employees, having a SQL or CSV format.

User permissions via *Azure AD Groups*. Synchronising user permissions with *AD Groups* permissions can eliminate the need to manually keep the user permissions, within the product, up-to-date.

Editable user permissions. Users with certain (administrative) permissions should be able to edit permissions of other users.

Could have

Employee recommendations for projects. An interesting extension (proposed by the company) is to incorporate a recommender system. This could, for example, suggest certain employees to work on a new project, based on their interests or past experience.

Integrate client data. Integrating client data adds useful extra features to display in the product, but since this information is confidential it is unsure whether we can acquire permission to view and process this data.

Virtual Reality (VR) traversal of visual graph. The use of VR is a nice-to-have. However, the company does not provide VR-devices for their employees by default. The implementation of VR requires a lot of effort while only yielding a 'gimmick'. Because of this, it is classified as a *could-have*.

Won't have

Data source: DMS. Although the stakeholder proposed the DMS as a data source, there are plans to replace this system in the near future. The new system is not likely to be similar to the old one, which is why it would be illogical to include this in our product. However, if we meet the requirement of having a generic solution (see page 3), the new system can be added afterwards.

2.3.2 Technical requirements

Must have

Clear, up-to-date documentation It is important that *Company X* is able to continue development of the product after the BEP. Therefore properly documented code is a must.

Should have

Support for Internet Explorer 11 (IE11). The majority of *Company X* employees use IE11 by default, but all employees have access to the *Google Chrome* [3] browser. Having to develop for a browser which lacks support for many Cascaded Style Sheets (CSS) and JavaScript functionality might cause the need to either create workarounds or compromise the quality of the end product.

An automated code checker for code formatting. This encourages the development team to write code according to a defined and well-structured format, which helps in future maintainability.

Easy-to-maintain codebase. A low code coupling and more generic methods can reduce bugs and allow both us during the BEP project and the company after the handover to add or change functionality more easily.

Unit tests. To gain confidence in the correct functioning of the product, each unit should be tested to confirm the behaviour is as expected.

Integration tests. To gain confidence in the correct functioning of the product, the units should be tested together to confirm that their combined functions are working as expected.

Continuous integration for automated testing. A continuous integration pipeline allows for efficient regression testing.

A minimal code coverage in tests of 80%. A higher code coverage[3] ensures that the tests cover a greater portion of the code of the product. However, this does not say anything about the quality of the tests. Making sure each test we have covers the entire functionality, seems more lucrative than making sure everything is tested. Especially as for some code it costs a lot of time to create the tests and the resulting tests would be highly complex. When there is a change in functionality of the code, it would really be a burden to maintain these complex tests. For these reasons we have decided that 80% is reasonable because this does ensure that a satisfactory amount of the code is covered by tests, but more importantly, the quality of the tests will be better as well.

[3] Code coverage is a metric to describe to which degree a code base is tested. The testing framework automatically determines whether each line of code has been run in tests and calculates the total coverage (as a percentage of the total number of lines of code).

Could have

Application: responsive design. Multi-device support is a nice-to-have, but we expect that the majority of users will interact with the product on a desktop computer or laptop. Therefore, responsive design does not yield a significant benefit. (See Appendix I).

Add or edit data via the application. The systems providing the existing data can be used to add or edit data if needed. However, if possible, it would be useful to be able to modify data from within the application.

Won't have

We do not have any technical won't-have requirements.

3 | Solution

This chapter will elaborate on *why* the solution was implemented and how the provided solution solves the proposed problem (section 3.1). The approach to the problem is outlined in section 3.2. Any deviations from the original requirements, made during the course of the project, are justified in section 3.3.

Finally, the impact of the product on day-to-day business will be discussed (section 3.4).

3.1 The product

The final product, a web application for internal use by *Company X*, visualises the relations between (groups of) employees, innovative projects they work(ed) on, and other similarities between them in an interactive network. This enables the user (a *Company X* employee) to answer questions like 'Who works on innovation?', 'Where is this happening?', but also find implicit relations like 'Which offices or teams of employees are often concerned with innovation?' or 'Which employee would be suitable to conduct a specific project'. A generalised approach of the implementation makes the product usable on other collections of data[1].

Visualisation

The application displays relations between employees, projects and groups of employees by means of a network of nodes, connected to each other with links, when a relation between the nodes exist. Several options for exploring, searching, compressing and filtering the network are provided, to reduce the amount of data shown and be able to intuitively answer the aforementioned questions. Figure 3.1 demonstrates the use of these tools to query the data.

Furthermore, the data shown in the network is configurable by admins and provides a solution which helps an admin deal with duplicate data records.

As the product visualises data in a generalised manner, and makes the data interactive, searchable and filterable for users, *Delphi* solves the problem as described in chapter 2.

3.2 Process

The whole project, from getting acquainted with *Company X* as a client and employer, to finishing the product and delivering this report, consisted of several distinct stages.

First the *research phase* served to formulate the requirements, explore possible solutions to several aspects of the challenge and, based on those findings, make several design/implementation choices (i.e. programming languages/frameworks, visualisation library, etc.). The challenge, research and choices have been documented in the *research report*, included in Appendix I.

During this research phase and before starting on the implementation of the product, the team had several meetings with *Company X* to ensure that they were on the same level of understanding. This way, the team could develop every stage, while keeping in mind what the final product had to look like. It was particularly important to prepare for the generalisation (data-agnostic nature), which has been an important and challenging requirement throughout the project.

The development phase was then started by making a Proof of Concept (PoC), which is a minimal version of the product to illustrate whether the design of the solution is suitable for the intended goal. As this simple version was as *Company X* expected, the team continued with the Minimal Viable Product (MVP). This version of the product includes all of the must haves (subsection 2.3.1), except the links to existing data (SP as a source, and AD for authentication) because these were not yet made accessible by *Company X*. It did, however, include a number of should haves already, including a start on the generic solution. This version functioned as the foundation for the final product, *Delphi*.

[1] The application can also be employed to show relations between other types of data, e.g. case laws and documents, or other relations between employees, like interests, hobbies, etc. The generic nature will be discussed further in chapter 4; Implementation.

[2] *Note: as Company X company data is confidential, we seeded our databases with fake data. This means that the illustrations in this and following sections show user interface components containing randomly generated data. Therefore, the data shown in illustrations can not be used to extract information about the company and any resemblance to actual persons is coincidental.*

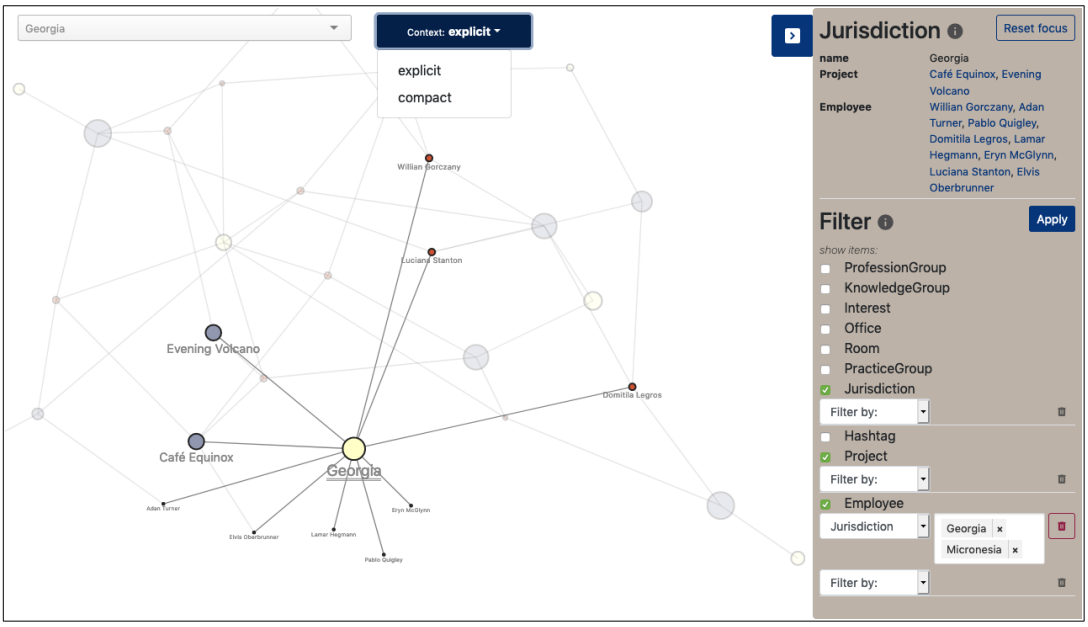


Figure 3.1: *Delphi* in action. Filters are applied to narrow the scope of the displayed information (to only include nodes related to *Jurisdiction Georgia*) or *Micronesia* and a node of interest (*Georgia*) is highlighted to emphasise its relations[2].

3.3 Deviations on requirements

Unfortunately, as in every project, things happen that require unexpected adaptation to certain constraints or requirements. This has also been the case in this project. This section will explain the deviations made to the original requirements described in section 2.3.

Shortly after the requirements were approved by *Company X*, it became clear that the "visual graph linking projects to users" (section 2.3) should have been a must have as this is the main part that could solve the problem defined in section 2.1. The absence of this visualisation would highly and negatively impact the usability of the application for *Company X*.

Linking to existing data

Linking the product directly to existing data sources did not go as expected. As *Company X* is, understandably, very protective about their employees' and company information, the acquirement of the access to this data was delayed a few times. In the last few weeks of the project, it became apparent that for the space where the application is intended to be deployed, the Microsoft Azure cloud, it will not be possible to have direct access to the locally hosted databases without decreasing the integrity of their data security. After some discussion, the decision was made that the company would still favour the product being a Software as a Service (Saas) application. Thus, the most favourable solution was a script, run internally, that exports data to the application periodically. The team implemented the possibility to import 'Comma-Separated Value' (CSV)-files. Because most software programs allow their data to be exported to CSV, including the desired SP data, this enables *Company X* to import the data into the application. Having made this decision with consultation with *Company X*, the proposed solution will meet the clients requirements concerning the import of real data.

Active Directory

The access to the AZURE AD user permissions suffered the same problems as acquiring the real data. As an alternative, a standard user-based authentication will be implemented. Admins will be

able to import data and alter permissions of other users and admin panel settings, while plain users can only view certain collections of data. At the time of writing, this feature is not yet present in the product, but it will be before the final version is delivered to *Company X*.

Extra features

As some of the requirements could not be met, due to the inaccessibility of real data, time was invested to implement other features. After a few meetings with *Company X* they expressed the desire for a filter (section 4.1.1) and a search (Figure 4.1.1) functionality. For this reason, these features have been added to the requirements and were implemented.

VR traversal of visual graph

Finally, the VR traversel requirement was deemed to be a stretch goal, that would have been a complete BEP-project of its own. For this reason it has been moved to the won't haves.

3.4 Impact

The product was designed to show relationships between entities in the data in an intuitive manner. It can be expected to increase the productivity of the employees at *Company X* by allowing them to access the right information faster and with less frustration.

Without this product, employees will need to use multiple software programs to find relations between data from different sources. This takes a lot of time and discourages utilising their data with full potential. As a complete overview of the projects at *Company X* and the employees that work in these projects does not exist, knowledge between similar projects is difficult to share. Contacting people about the status of projects is also harder to do.

Without an overview of past and current projects of employees, chances are greater that employees assigned to new projects will not be the best possible choice. This leads to projects that do not maximally use employees' knowledge, interests and experience, which can make the projects take longer to finish and get more expensive than necessary.

4 Implementation

This chapter provides an in-depth technical analysis of the final product. The implementation details will be discussed in section 4.1, while efforts to ensure code quality are outlined in section 4.2.

4.1 Final product

In terms of implementation, the product can be split in two distinct parts. The back-end, responsible for managing and delivering the data in the right format, is expanded on in subsection 4.1.2. The front-end, responsible for displaying the data provided by the back-end and processing user activity, will be further explained in subsection 4.1.3. In general, the front-end requests data from the back-end, based on user input (e.g. clicking buttons, typing a search query) and the back-end responds by providing suitable data for this user input.

4.1.1 Features

The final product can be split into its most important features. Every feature will be described in order to explain its functionality.

Network

The main feature of the product is the network. In this network the imported data is visible as a network of nodes with links in between. These nodes represent the entities present within the data, and links represent existing connections between these entities. The nodes can be clicked and dragged to alter the visualisation. The nodes have labels containing their names, and have sizes proportional to the number of links. The nodes space themselves away when they are too close to other nodes to keep the structure of the network as uncluttered as possible.

Focus

Focus can be applied by double-clicking a node. This will place the node in focus, which visually translates to the following sequential actions: centering the node, highlighting the node, playing an animation to center attention on this node and showing detailed information about the node in the Sidebar. From this point the user can start highlighting other nodes.

Highlights highlight the selected node to make them visually easier to see. The user can trigger a highlight by single-clicking the desired node. The highlighted sub-graph will consist of the clicked node and its first degree neighbours. The user can, if he/she wishes, click on these neighbouring nodes to continue highlighting its neighbours as well. Finally, the user can exit a highlight by single-clicking the same node again. In this manner, the user can traverse the graph by using highlights.

Focus animation When a node is focused, an animation is played on the node to visually assist the users on which node they clicked. Currently, the animation consists of the enlargement and shrinking down of the node twice. As a lot of nodes can be present in the graph, this animation creates awareness to the user which of the nodes in the graph they have just selected. Besides that, the label under the focused node is underlined, so after the animation, the user can still see which node has the focus.

Context switch

The context switch is a button found next to the Searchbar. It allows the user to switch to a different graph visualisation. Currently, there are two different contexts:

Explicit is the default graph visualisation (example can be seen in Figure 3.1). By selecting two different data types, e.g. Project and Employee, the graph will show links that indicate direct relations between these types, i.e. A link between a project and an employee tells the user that an employee works on a project.

Compact shows the implicit graph visualisation. Once again, if we take the data types Project and Employee as an example, links show the implicit relationship between these types, i.e. A link between employees tells the user that these employees share at least one common project. This is illustrated in Figure 4.1. By switching the order of data types, one could

also show a graph where projects share at least one common employee.

Searchbar

The final product allows the user to search a desired node in the graph. The user can type search strings, e.g. names of employees, in the top left corner of the screen. This Searchbar can auto-complete the search string in case that the user does not know the entire name. The user is able to search for one item at most at a given time.

The Searchbar also functions as a drop-down menu. Here, the users can find all the nodes currently present in the visualisation. All the nodes are categorised in the drop-down list with their matching data type, e.g. Projects, Employees or Jurisdiction.

The graph will put the desired node after selecting in focus.

Sidebar

The users sees a combination of useful features at the right side of the screen. The user can fold this sidebar to the right, to free up more space on the screen to show the network. In addition to showing detailed information about a node after it is put in focus, the Sidebar houses several components to assist the user in the visualisation.

Including or excluding types

In the sidebar, checkboxes can be clicked by a user to select different data types to be shown in the visualisation. By clicking the checkboxes found in the Sidebar, the user can add or remove data types to the graph.

Detailed information

Also situated in the Sidebar, the detailed information of a node is shown whenever the user has put the node in focus. The attributes that are displayed within the Sidebar vary depending on the data type and is adjustable in the admin panel. When users find the detailed information insufficient, they can click on a button that opens a page with all available information about that node on a new page.

Filters

Filters allow the user to adjust the visualisation based on different attributes of the selected data types. For each data type, a filter bar is shown in the Sidebar. Per type, a filter can be applied, and multiple filters can be active at once. Each filter reduces the number of nodes of the corresponding type visible in the graph, depending on the filter criteria. Text filtering can be applied per attribute (e.g. Employee nodes with a name containing the text 'Eve') and per link (e.g. Project nodes that are linked to Countries The Netherlands, Belgium and France).

Admin panel

The application contains a admin panel. In the admin panel there is a page for selecting which attributes and relations should be hidden in the detailed information panel within the visualisation. Additionally, pages for importing records from CSV files and for showing and merging possible duplicate nodes are present. Detailed explanations of these functions can be found in subsection 4.1.2, starting from page 14.

4.1.2 Back-end

The back-end handles all the data storage and manipulation, and serves HTML pages to browsers. It is responsible for importing data from external sources, showing the information pages of nodes, transforming node data to be able to render the network and the functionality of the filters within the network.

Ruby on Rails

The back-end of the application is implemented in Ruby, with the Rails framework for database management and serving web pages. By default, Rails installs with *Minitest*[1] and *SQLite3*[2], which we use for testing and database storage. Rails supports multiple environments which have separate configurations and databases. We use separate environments for development, production and testing. This prevents unwanted data to end up in the wrong database.

[1] https://github.com/seattlerb/minitest
[2] https://www.sqlite.org/version3.html

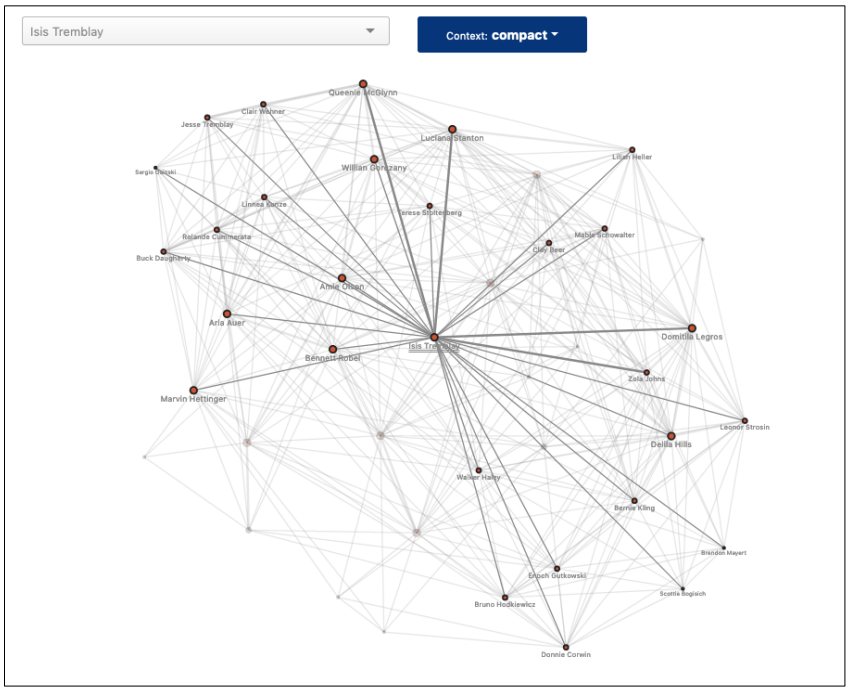


Figure 4.1: Compact context. This context shows links between employees, where the width of the links represent the number of projects they have in common.

Database tables and models

In order to make the product generalised and 'data content agnostic', the database and application are setup in a way to make as few assumptions over the contents as possible, while maintaining its usability. This has the benefit that this product can be used in different domains.

Within the Rails framework the database tables are paired with Rails models, so the relations between models are representative of the relations within the database. The relations between the models are visualised in Figure 4.2. The figure shows the existing models with filled arrows representing 'belongs to' relations, and grey boxes and arrows representing virtual models and relations. Modules are not shown in this figure as they are not models, but rather extensions of models. These models are explained in more detail below.

DataTypes represent types or classes. Examples are `DataTypes` with names 'Project' or 'Employee'. `DataTypes` can have multiple `DataNodes`.

DataNodes represent the records or entities of a certain `DataType`. A `DataNode` has a name and can have `DataAttributes` and `DataLinks`. A `DataNode` is converted to a node in the network.

DataNetworkable is a module that the `DataNode` model includes. It provides methods that are useful for building the data needed for the front-end to draw the network.

DataMergeable is a module that is also included by the `DataNode` model. It provides methods that are used for comparing and merging `DataNodes`.

DataLinks represent the links or connections between two `DataNodes`. While the model acknowledges a source and target `DataNode`, within the application `DataLinks` are treated as bi-directional. `DataLinks` can be converted to links between nodes in the network.

DataAttributes represent the attributes or characteristics of `DataNodes`. A `DataAttribute` has a `DataAttributeKey` and a `Valuable`.

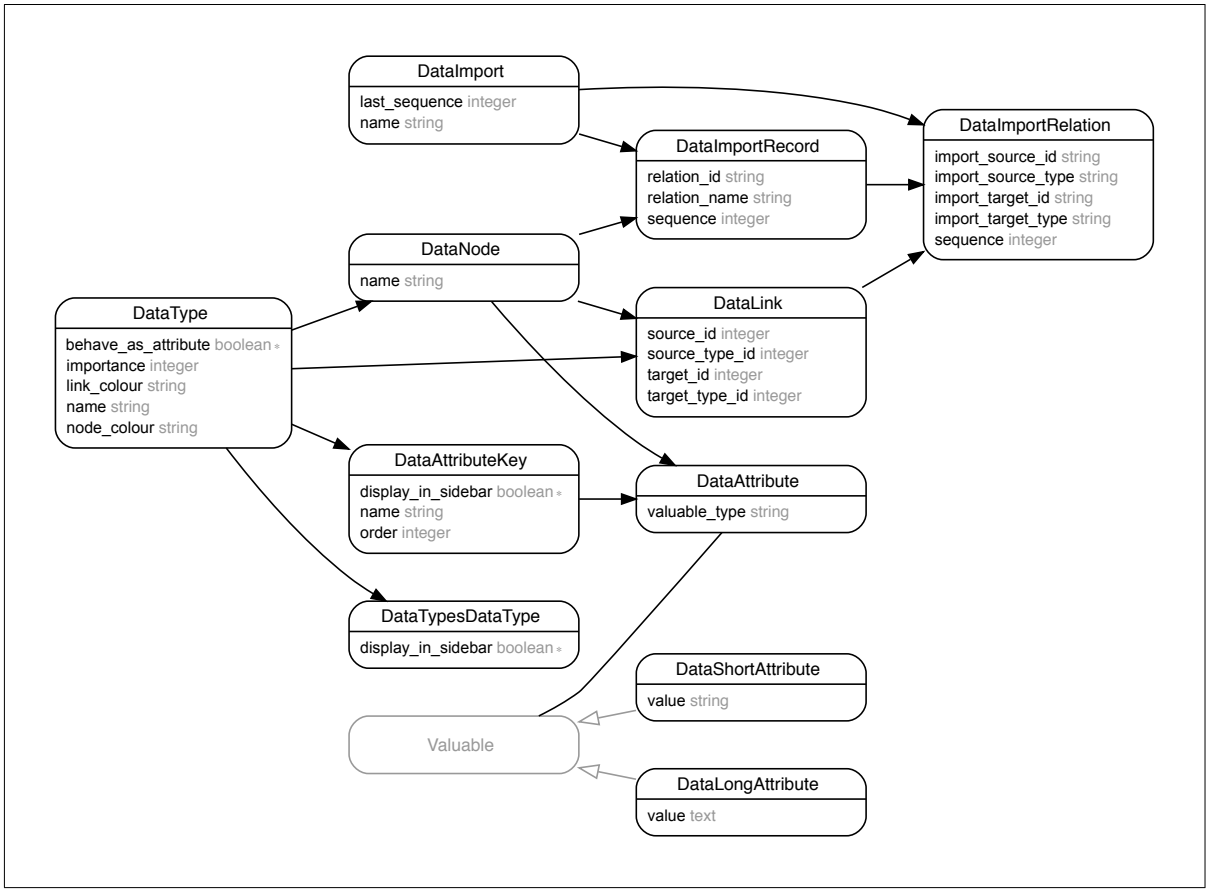


Figure 4.2: Entity Relationship Diagram of generic models

DataAttributeKeys represent the names of attributes that DataNodes can have. Examples are 'description' and 'email'.

Valuables are virtual objects that display polymorphic relations between DataAttributes and can either be DataShortAttributes or DataLongAttributes. Polymorphism in this context means that a DataAttribute does not have to define beforehand whether it has a DataShortAttribute or a DataLongAttribute and can switch between them.

DataShortAttributes and DataLongAttributes are concrete models of the virtual model Valuable and represent the values corresponding to the DataAttributeKeys. If the value of the attribute has a length greater than the limit of a string column in the database (255 characters), it is stored as a DataLongAttribute with a text column (limit: 65,536 characters), otherwise it is stored as a DataShortAttribute. The distinction between DataShortAttribute and DataLongAttribute was made to allow storing large sized attributes, while saving disk space on smaller sized attributes.

DataTypesDataTypes represent the current connections between DataTypes. Every time two DataNodes are connected via a DataLink, the corresponding connection between their DataTypes is created if it does not already exists. Additionally, admin configurable preferences for presenting the data in the network are stored within instances of this model.

DataImports are used to store metadata about imports. A DataImport is used to determine whether new DataNodes should be created or existing DataNodes should be updated during an import.

DataImportRecords complement DataImports with information about the type and ID of a record in the import data.

DataImportRelations complement DataImportRecords by providing information about connections between records in the import data.

Generating test data

As it has not been possible yet to use the application with real *Company X* data, a data generator was implemented. This generator fills the database with dummy data, which was useful during developing and testing. The generator can be configured with parameters to determine how many DataNodes per DataType should be generated. The generator can be run with one console command to fill or refill the database.

Network API

To draw the network, the front-end needs data from the back-end. This data can be obtained via API end-points. The front-end accesses four end-points, each one of these has a separate responsibility.

The first end-point is called when starting the visualisation in order to know what DataTypes are available and for which ones the nodes and links should be displayed by default. Additionally, per type, the attribute names and associated types that can be used for filtering are provided.

The second end-point provides the nodes and links in JSON format that are needed to draw the network. This end-point accepts a list of DataType names and, optionally, a list of filters. These parameters are then used to build the data for the network. Whenever a user includes or excludes DataTypes to be displayed, or has used the filters, a new request is made and the network is rebuilt.

The third end-point is used to provide data which the Searchbar in the front-end uses to make relevant suggestions, based on which DataTypes the user has selected and their keystroke.

The last end-point is used to load the detailed information of a DataNode and is requested when the corresponding node in the network is in focus. This includes its attributes and connections to other nodes. The application takes into account the configured preferences defined in the admin panel and shows or hides attributes and relations based on this.

Generating the network

The network is generated using DataNodes and DataLinks. From these, the records from the included DataTypes are selected from the database. These records are then converted to lists of node

and link representations. For the compact context, the number of common connected nodes for each pair of DataNodes is used for the link representations, instead of the DataLinks.

Configure data presentation in network

The admin panel contains a page where certain aspects of the visualisation are customisable. Each DataType can be configured to be excluded from displaying nodes in the network. Per DataType, each attribute and connected type can be shown or hidden in the detailed information.

Import

The import page can be used to import records from external sources. We have chosen importing via CSV files, because access to the *SharePoint* databases containing the real data is not possible from the cloud where the product will be deployed. CSV files are relatively easy to create or tweak and CSV exports are a common way to export data in software programs. This decision has been made in deliberation with *Company X*, who indicated that this solution fulfils their needs.

The import page asks for a CSV file. When uploaded, the file is parsed. The filename is used to extract the name of the type. The first line in the file is used for extracting the attribute names. Columns with the suffix '_id' or 'Id' are assumed to be foreign keys of a type that corresponds to the substring before the suffix.

The user has to provide an import name, which will be used to link the data to DataNodes or DataLinks that have been created with the same import name. The user can change the inferred attribute names and types or exclude them before the data is imported. Additionally, the user can mark the file as containing a join table, after which the import only creates DataLinks.

On each import, connections with non-existing DataNodes are stored, so that when the corresponding DataNode is created in a later import, a DataLink between these DataNodes will automatically be added afterwards.

Duplicate nodes detection

When importing from multiple sources, it may happen that two or more DataNodes represent the same entity. For the application it is not possible to detect with perfect accuracy which nodes are duplicate nodes. The application can, however, suggest possible matches based on the similarity of their names.

As string comparison for every DataNode with every other DataNode can be time-consuming for a large number of DataNodes, the page by default loads possible duplication matches for the 250 most recently added DataNodes. The page also supports loading matches for DataNodes with names starting with a given letter.

Although merging DataNodes can be done directly in the duplicate nodes page, it is also possible to go to the compare page before merging. This page can also be used to compare and merge DataNodes that were not suggested by the application.

In order to quickly find DataNodes with similar names an algorithm was implemented that can scale by trading off accuracy for speed. In short, the algorithm takes small random substrings from one name and checks whether this substring is present in the other name, and repeats this. The ratio of matches per number of substrings gives a similarity score. Due to the random nature, few repeats cause the output of the algorithm for the same names to fluctuate every time it is run. With a larger number of repeats the output will fluctuate less and converge towards a certain ratio, according to the law of large numbers[3]. If the outputted ratio is above the acceptance ratio, it will flag the two nodes as similar. By tweaking the acceptance ratio, the comparison will vary in strictness. The pseudocode for this algorithm can be found in Figure 4.3.

4.1.3 Front-end

The front-end is responsible for rendering the network and providing a Graphical User Interface (GUI) that the user can interact with.

Structure

To develop the parts interacting directly with the user (i.e. the front-end) of the application, the *Vue.js* framework was used. This is a popular JavaScript (JS) framework that shares a lot of features with the

[3] https://www.britannica.com/science/law-of-large-numbers

```
function compare_node_names(name_1,
    name_2, cycles, accept_ratio) {
  if name_1.equals(name_2) {
    return 'identical'
  } elsif (calc_similarity(name_1,
      name_2, cycles) >= accept_ratio
      ) {
    return 'similar'
  } else {
    return 'different'
  }
}

function calc_similarity(name_1,
    name_2, cycles) {
  count := 0
  min_size := 2
  max_size := max(2,
    min(log(name_1.size), log(name_2
        .size)))

  for i in 1:cycles {
    names := shuffle([name_1, name_2
        ])
    name_this := names.pop
    name_other := names.pop

    sample_size := random(min_size,
        max_size)
    sample := random_sample(
        name_this, sample_size)

    if name_other.contains(sample) {
      count := count + 1
    }
  }

  return count / cycles
}
```

Figure 4.3: Pseudocode of algorithm to determine similarity of two node names

React-framework (which serves the same purpose). The application uses *Vue.js* SFC, which organise the HTML (layout), CSS (styling) and JS (behaviour) belonging to a single *component* in a single file[4]. A *Vue.js* component can be re-used multiple times and can counteract writing large amounts of similar code.

As stated, *Vue.js* components are deeply nested and are used to build modular parts that are contained in parent components. Figure 4.4 demonstrates this by means of a hierarchy/dependency diagram. Each green block in the diagram represents a module, and the arrows in between nodes show that one component is situated inside the other component. Each white block represents a group with components that share similar functionality, the more groups two components have in common, the greater the similarity of their functions. The following sections will go in-depth on the different components developed and used in the application, and the style and design choices.

***Vue.js* components** First, a little background on *Vue.js* is useful. Components can be used as a top-level component (i.e. 'main screen'), but they can also be nested within other components. In practice, this happens a lot. Components can pass data to child components via *properties* (e.g. the `value=""` known from plain HTML. A component can send data to its parent using *events* (example from standard HTML: `onclick(data)`, an *event* which is fired whenever an element receives a mouse click). The parent component can listen for specific events and handle them.

This section explains the layout and logic of every component (i.e. user interface building block).

Network The `Network` component is the top-level component. It is essentially the main page of the application. It houses all other components of the application but it consists largely of the HTML `<svg>` element used to draw the visualisation on. Besides, it houses the logic for the visualisation as well as the main control logic (perform data

[4] Not using Single File Components (SFCs) is possible, but disables some features the product depends on, like: per-component styling, `Babel` JS pre-processing (necessary for IE11 compatibility, and several development benefits.

[5] `https://github.com/sverweij/dependency-cruiser`

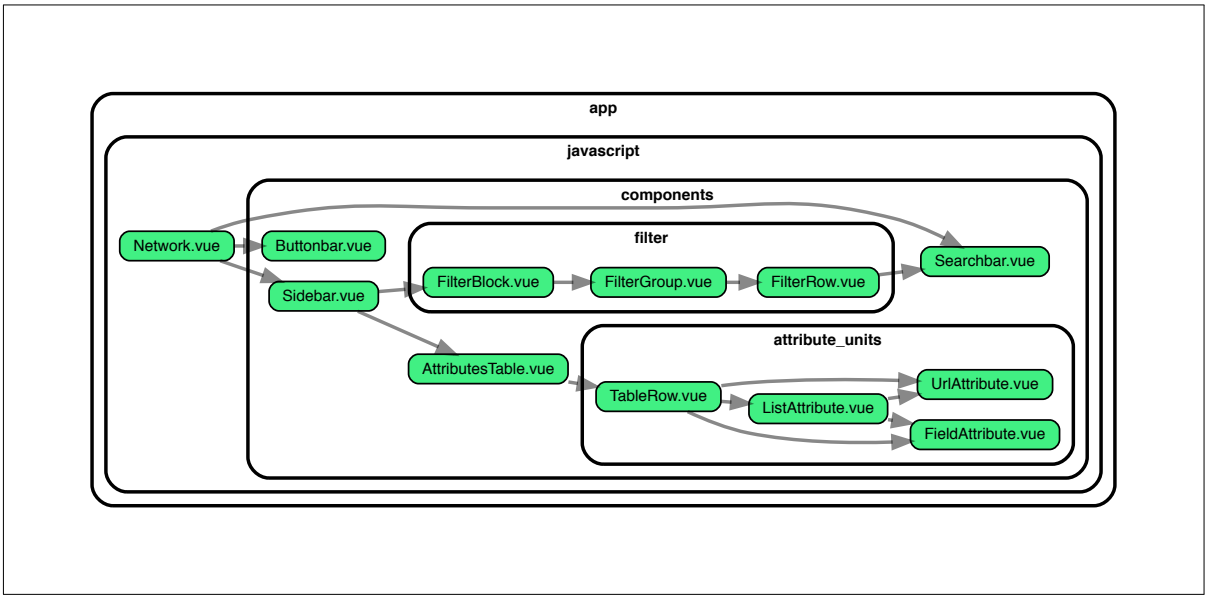


Figure 4.4: A diagram of the relations between all *Vue.js* components. It shows the hierarchy further described in subsection 4.1.3. *Diagram created by Dependency Cruiser*[5].

requests, orchestrate control of all components and keeping client side data).

Apart from the `<svg>` covering the whole screen, the `Network` displays a `Searchbar` and `Buttonbar` at the top of the screen to control and navigate the visualisation and a collapsible `Sidebar` on the right to display textual data and house more controls.

Searchbar During the course of the project the need to search for specific node arose (see Figure 4.1.1). This component offers the possibility to search for known data. It is implemented in a versatile manner, to be used in multiple roles.

It is used for the main searchbar, always visible at the top of the screen, but is also used in the `FilterBlock` to filter on type relations.

The `FilterBlock` searchbar can select multiple nodes of the same type. In this case, every selected node will show as a little block (for an example, refer to Figure 4.11). In this case, all selected elements are added to a list. Every change emits an *event* to the parent `FilterBlock`, which inserts the list into the filter query.

The main searchbar only allows a single node to be selected. Therefore, instead of using the selection boxes, the placeholder of the searchbar shows the *name* value of the selected node. This is a more natural way to show a single selected element, as shown in Figure 4.6.

In contrast to the `FilterBlock` searchbar, the main one displays multiple types (namely, all enabled/shown types). The nodes are separated by type using option groups, as shown in Figure 4.5.

Buttonbar The `ButtonBar` is set up in a generic way. It can be used for multiple sets of options. In the application it is used to select the visualisation *context* (ref). A list of properties is passed in as a *property* (refer to Figure 4.7 for an example of the result) and upon a click, the component emits an *event* with the selected option as data. In this case, this triggers a context switch (introduced in subsection 4.1.1) API request in the `Network` component.

Sidebar The `Sidebar` is mainly a container to layout multiple child components. It houses logic and styling to collapse itself, besides a `FilterBlock` and an `AttributesTable` (example showing both elements in Figure 4.8). The latter is only displayed when the `Sidebar` is passed a node *type* and *ID* to show the data for, which happens when a node is in focus.

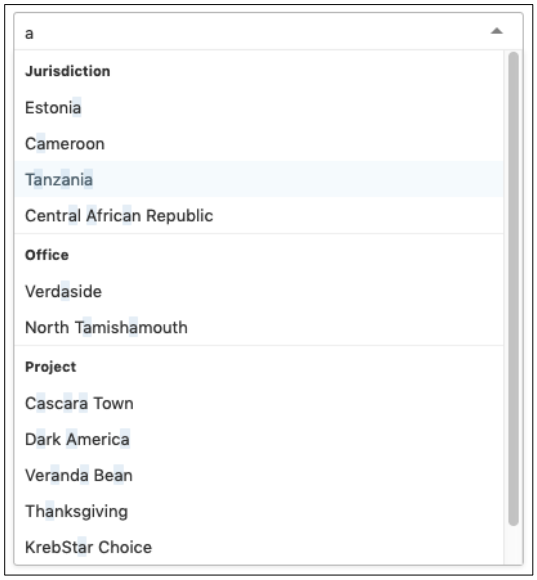


Figure 4.5: The main searchbar, open when typing a query. It shows all visible nodes, grouped by type.

Figure 4.6: The main searchbar uses a placeholder to show the name of the selected node, providing a natural, discrete way to display a single selected value.

Figure 4.7: The `ButtonBar` shows the different context options, passed from the back-end.

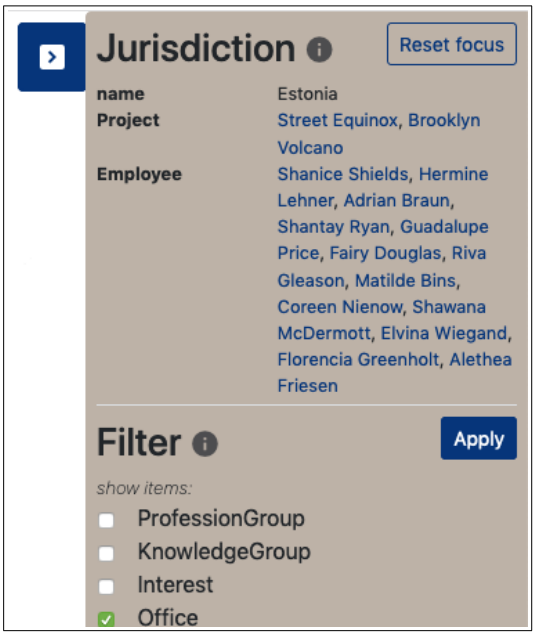


Figure 4.8: The `Sidebar` shows an `AttributesTable` (but only when a node is in focus) on top of the `FilterBlock`.

AttributesTable Designed to display specific attribute/type (see subsection 4.1.2) key/value pairs of a *node*, this table just creates a multitude of `TableRows`; one for every item in the list of key/-value pairs it receives as a property. The back-end serves the data in the following generic and versatile format shown in Figure 4.9.

In Figure 4.9 is a sample of the data that resulted in the `AttributesTable` as shown in Figure 4.10. Three types of keys to show in this table exist; `field`, `list` and `url`. This key is not explicitly displayed, but controls how the *value* of the attribute is shown. The three types will be further explored in the corresponding sections below.

TableRow One *key/value* pair in an `AttributesTable` (for an example, refer to Figure 4.10). Simply displays the *key* in bold font and the value next to it, unless no *key* is present, which will result in the value being spread over the whole width of the row. This is useful to display long stretches of text (e.g. a summary, which does not need a *key* to explain its meaning).

The value displayed in the `TableRow` can

```
[{
  id: 0,
  type: "field",
  key: "name",
  value: "Joe Delight",
  target_id: null,
  actionable: false
}, {
  id: 1,
  type: "list",
  key: "Employee",
  value: [{
    id: 0,
    type: "field",
    key: "Kim Bogan",
    value: "Kim Bogan",
    target_id: "Employee-2576",
    actionable: true
  }, {
    id: 1,
    type: "field",
    key: "Leo Langosh",
    value: "Leo Langosh",
    target_id: "Employee-2584",
    actionable: true
  }]
}]
```

Figure 4.9: Sample of the data used to populate the AttributesTable.

name	Joe Delight
description	faint, coating, dates, sage, milk chocolate
archive	current
Jurisdiction	Guinea
Employee	Kim Bogan, Leo Langosh, Galen Spinka, Susan Waters, Terry Heathcote

Figure 4.10: An example of an `AttributesTable` showing details of the Joe Delight project.

be one of `FieldAttribute`, `ListAttribute` or `UrlAttribute`.

FieldAttribute Used to either show a text value in an `AttributesTable`. It can either be *actionable* (in essence: clickable) or not. Clicking it results in an *event* (`action(target_id)`) being emitted to be handled by `Network`. In this case, this is used to focus a node after it is clicked in the sidebar.

ListAttribute Used to show lists of `fields` or `urls` (or even mixed). It receives a suffix (default: ',') which it passes to its child components (unless they represent the final element). This results in the list as shown in Figure 4.10.

UrlAttribute Used to render an HTML `<a>` (link) element with custom text value. It is implemented in the current version of our application, but will be shown if and only if the back-end sends `url` type data to display in the sidebar. At the moment this is not used. It was implemented because we suspect this will be needed when using the application in a different domain.

FilterBlock This component is the highest level component responsible for composing a filter query for the visualised data (i.e. all other filter-related components are children of this one). An example of this component in use, outlining the component hierarchy, is in Figure 4.11. This figure shows the `FilterBlock` with some filters in place; the *Project* and *Employee* data types are enabled, so only those will be shown in the visualisation. The *Employees* are being filtered on two criteria; their *name* has to contain 'john' (case insensitive) **and** they should be related to *Office* 'Kunzeville'. Note that a node has to satisfy **all** filter criteria for its type. A `FilterBlock` employs `FilterGroups` to do the heavy lifting in user interaction and assembling the filter query from the input.

FilterGroup A group of filter criteria (i.e. `FilterRows` for one type of nodes. The `FilterGroup` component houses most of the logic to assemble a filter query, which is a representation of the user input.

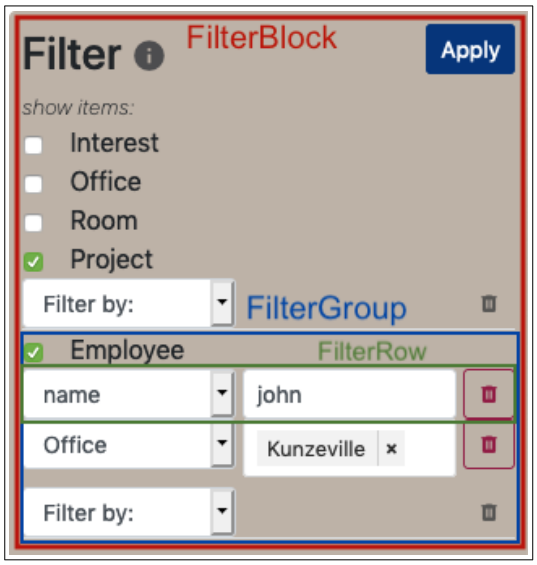


Figure 4.11: The component hierarchy of FilterBlock (responsible for the complete filter), FilterGroup (represents a filter for a single type) and FilterRow (a rule to filter on an attribute or type).

In order to save the user from unnecessary clicks, the `FilterGroup` manages adding/removing `FilterRows`. It shows no rows when disabled, otherwise it shows one. Adding rows by hand is not possible (and not necessary), as a new empty row is added when the previous one is filled in. Deleting rows by hand is possible, in case the user needs to slacken the filter.

FilterRow A `FilterRow` represents a single attribute *key/value* pair in the filter. It includes a HTML `<select>` element to show the types/attributes to filter on. They are in a single list because in practice, the user should not be concerned about the difference.

Choosing an attribute will show an `<input>` element. In this case, the entered value will be a string,[7] which will be passed to the parent `FilterGroup` by means of an *event*, where it will be processed.

If the user selects a type instead of an attribute, a `Searchbar` is shown. In this case, it shows all nodes of the selected type and allows the user to search and select multiple items. This results in a list of node *IDs* being sent to the parent component.

Style and design

An important way to ensure usability of an application is to provide an attractive design. Furthermore, usability can be improved by making it responsive (i.e. adaptive to different screen sizes/ratios). As already established during the research phase (Appendix I), *Bootstrap* is a straightforward CSS and JS framework to provide a responsive layout as well as an attractive, modern design. It provides a grid system[6] to easily arrange HTML elements, which is the base of the layout of the *Delphi* application.

Another benefit of *Bootstrap* is the possibility to 'theme' it[7]. The *Company X* corporate identity colors act as as primary colors in the application and *Bootstrap* adopts them seamlessly. Therefore, the application style is consistent.

4.1.4 Visualisation

The visualisation is a big part of the front-end display that users will see and interact with. D3.js[8] is a JavaScript based library. Its main purpose is the manipulation of data into an interactive visualisation using HTML, Scalable Vector Graphics (SVG) and CSS. D3.js achieves this by allowing you to bind arbitrary data into the Document Object Model (DOM), and then transform the object by data-driven transformation.

D3 is capable of handling large amounts of data as it only modifies the attributes that actually change. However, visualising everything at once will reduce the responsiveness of the browser. Reports on D3's visualisation performance differ[9,10]. However, for a visualisation to be clear and legible, it should not contain too many elements (in the order of hundreds). Therefore a solid dynamic (depending on context and number of selected data points) server-side abstraction is essential.

In our application, D3 is used to visualise the network that visualises the data and their relations

[6] `https://getbootstrap.com/docs/4.0/layout/grid/`
[7] `https://getbootstrap.com/docs/4.0/getting-started/theming/`
[8] `https://d3js.org/`
[9] Zach Nation. *3 Steps to Scalable Data Visualization in React.js & D3.js*. URL: `https://www.codementor.io/znation/3-steps-to-scalable-data-visualization-in-react-js-d3-js-8t7kjxnk5` (visited on 03/05/2019).
[10] *D3.js - Performance Test*. URL: `http://tommykrueger.com/projects/d3tests/performance-test.php` (visited on 03/05/2019).

as mentioned in section 2.1. This network consists of the following three entities:

Nodes represent the different entities in the graph visualisation. The size of the node is determined by the number of connections it has with other nodes.

Links represent relationships (e.g. group membership) between pairs of nodes.

Labels are used to show the names of the nodes.

4.1.5 General

This section serves to discuss some important, high level properties about the product.

Generality

The application was built with generality in mind. Because of this, the application makes minimal assumptions about the structure of the data. This has the result that multiple copies of the application can visualise different types of data with no adaptations in the code. It is also capable of extending on already imported data with new data sources. As most programs can export their data to a CSV file, exporting data from existing sources should not be difficult.

Maintainability

As not much can be assumed about the input data, the source code, in both the back-end and the front-end, has to be highly modular. This has the advantage that adding or altering models, methods, components, database tables and even changing the type of database can be done without a lot of alterations in the code. Having several code branches, databases and back-end environments is also advantageous, because new features can be deployed and tested on a separate branch and environment, while the main application only contains well tested features on a separate database. This should minimise the number of bugs that users will see.

Scalability

Although generality comes with many advantages, it also has its drawbacks. Due to the structure of the models (see subsection 4.1.2) fetching all the detailed information about a single entity requires querying through six database records, instead of one or a few. This structure also makes it harder to efficiently search through the database for filtering. While the application can still be successful with building and rendering thousands of nodes and links, there is a noticeable delay between when the user performs an action, and when the application shows the desired response.

Compatibility

Browser support As the computers at *Company X* have IE11 as default browser and many employees use it, browser support has been an important part of the development process. Overall, quite a lot of time had to be spent in order to create the workarounds in order for the application to support IE11.

The most noticeable difference between our application in IE11 and other major browsers is the performance in terms of responsiveness of the visualisation, where IE11 is definitely the worst performing.

The final product does completely support all major browsers.

Screen size The application scales the user interface components depending on the width of the browser. This allows users with smaller screens to still be able to navigate the visualisation and use all of its features.

4.2 Code quality

As the product has to be used and maintained at *Company X* after finalising the project, it has been an important goal to deliver maintainable, well-documented code. During the project we experienced that it is not easy to keep tests as well as documentation up-to-date constantly. However, this section will present the result of the efforts made over the complete development phase.

4.2.1 Testing

Testing is an important part of code development. A high code coverage ensures that the tests cover a greater portion of the code of the product and

ensures that the functionality of the product is as expected. We have split the tests into two environments, as the product has a back- and front- end.

front-end

The front-end, developed in *Vue.js*, was tested using the framework designed specifically for this purpose; Jest.

As the front-end consists of small, reusable *Vue.js* components, each can be tested individually. This is used to ensure several aspects of the front-end functionality, including:

1. Whether the correct web request is performed, and whether the component acts upon the response data as expected.
2. Whether specific actions have the correct consequence, for example clicking a node to highlight or focus it.
3. Whether expected parts of the components (like specific buttons) exist, to prevent them from being moved or removed by accident.

To test the front-end of a constantly evolving product is not trivial because the components to test are rather big. This results in the coverage numbers as presented in Figure F.1. The four columns after the files' name indicates a measurement of test coverage. Then the following measurements are being collected:

Statements The statement coverage collects the percentage of executed statements in the code.

Branches Branch coverage aims to cover all the outcome possibilities at each decision point, e.g. if-statements. This ensures that the outcome is as expected.

Functions Function coverage collects the percentage of methods that are being tested.

Lines Line coverage is often the least meaningful measurement, as the number of statements on a line could be more than one. For example, imagine a line with two statements and only one of these statements are being tested. Then the corresponding coverage will be hundred percent line coverage and only fifty percent statement coverage. It is good practice to limit each line to one statement, resulting in that the percentage of statement coverage is often equal to the line coverage.

The combination of statement coverage and branch coverage would be an ideal measurement for the quantity of code coverage. As mentioned in the Appendix I, we tried to achieve a code coverage of at least 80%. Then the coverage report shows a satisfactory coverage of 83.5 % and 67.5 % for statement coverage and branch coverage, respectively.

Back-end

The *Ruby on Rails* back-end is tested using the default testing framework provided by *Rails*; *Minitest*[11]. This is easy to comprehend and powerful, which ensures testing is intuitive and quick.

The result of 30 tests (checking 100 assertions) is shown in Figure F.2.

4.2.2 Formatting

We have made use of *linters* to keep the format of the code consistent during development. A linter is a static code analyser. The developers have to abide by the format rules they set themselves. Keeping the format consistent contributes to the overall readability of the code.

Front-end

A popular linter for JavaScript is *ESLint*. ESLint is an open source linting utility.

The noticeable stylish rules that were applied are the following:

1. The placements of commas in JavaScript remained consistent.
2. The length of each line had to be kept under 80 characters.
3. The use of print statements, i.e. the use of console.log(..), was not allowed in the final code.

Back-end

We have utilised *RuboCop* as the linter for the back-end. In addition to the static analyse, RuboCop is also able to fix some of these format issues automatically.

Examples are:

[11] `https://github.com/seattlerb/minitest`

1. Limits for class, module and method line lengths.
2. Avoidance of double negatives in conditions (e.g. 'if object.present?' instead of 'unless object.blank?').
3. Encouraging using descriptive code (e.g. 'number.positive?' instead of 'number > 0').

4.2.3 Documentation

The application is described in the code documentation for the back-end and front-end, an entity relationship diagram for the existing models, a setup document and a hand-over document. Each type of documentation is described in the following sections.

Code documentation

Keeping documentation up to date can be cumbersome when working continuously on a product. As a solution, we have decided to automate this process. We use two libraries, *SDoc*[12] and *Vuese*[13] to generate documentation from comments in HTML format, for the back-end and front-end, respectively. Both documentations have a list of classes or components and a search bar to browse through, and a page for each item with its methods. Every time a change in classes or component is committed, the relevant pages will be regenerated.

While this approach allows the the documentation to be updated frequently, it does not guarantee that the documentation is up to date. Comments have to be added or updated when methods change. We strived to comment new methods whenever they were added and bi-weekly do a full documentation check.

Entity relationship diagram

Whenever the structure of the database is altered, e.g. by adding a table, an entity relationship diagram (example: see Figure 4.2) is generated and saved as Portable Document Format (PDF) file. This way, the diagram is kept up-to-date at all times. The diagram shows attributes of models and relations between them.

[12] `https://github.com/zzak/sdoc`
[13] `https://vuese.org/`

Application setup document

To get started with running the application a 'readme' file was included in the folder that contains the source code, that describes how to get the application to run. This includes installing the dependencies, setting up the database and deploying the application on a server.

Hand-over document

We have planned write a hand-over document in the last week of the Bachelor project to inform the company on how to maintain or expand the application after the Bachelor project ends.

4.2.4 SIG

This section will contain the feedback that Software Improvement Group (SIG) has given us about our code (in Dutch). The code has been submitted twice, once in week six (Appendix G), and once in week nine (Appendix H).

Feedback - week 6

The first evaluation of SIG was relatively positive with 4.1 stars out of five, which means the code scores above average on SIG's maintainability model. It mentioned two main issues, being some code duplication and a few methods that started out as small methods and since then have been extended, which caused these methods to be of above average length.

At the moment the team received the feedback from week six, the production code had already changed quite a lot, which actually solved the problems mentioned. Mostly because of the fact that in that week, a change was made from specific data sources to a more generic solution, which got rid of most of the duplicate code and made most of the code a lot shorter (less hard coded data types).

This means that the value of the feedback for the team is mostly a warning to not make these same mistakes again by keeping the suggestions in mind when adding new production code.

Feedback - week 9

The SIG feedback of week 9, which can be found in Appendix H, confirmed that we indeed improved

on the points mentioned in week 6. Besides that, they mentioned that the maintainability of the production code has improved, despite the fact that there is more production code. Also, they were happy to see that new test code has also been written. To conclude, the recommendations from the previous evaluation were included in the development process.

5 Research

5.1 Research Question

The question we seek to answer is ***"To what extent can the visualisations, i.e. explicit and compact, be intuitively grasped and provide the information they are supposed to provide?"***

In order to investigate this question, the following sub-questions need to be answered.

- When is a visualisation successful?
- How does the user experience the current visualisation?
- How can the result of the user study be used to improve the visualisation?

When is a visualisation successful? "For the transfer of knowledge in organisations interactive visualisations help for example to fascinate recipients, to enable interactive collaboration, to reveal unforeseen relationships and to present complex data."[1]

For *Company X*, a visualisation is considered a success when it reveals relationships that cannot be seen with their current systems. Our initial product (chapter 3) provides these relationships in a visualisation. However, the question is to what extent the end user is able to grasp this information.

In order to verify the success of the visualisation and to improve the product, the research question needs to be completed. Thus, a user study was set up. In this study, multiple employees of *Company X* tested the product by completing a set of tasks within the application and going through multiple visualisations. By tracking the different results (i.e. number of clicks, time, subjective opinions), the team was able to answer these questions.

5.2 User study

5.2.1 Setup

As people are alike in some ways, but different in others, every new participant finds a lot of the same problems and some different ones in the product than their predecessor. The usability problems found will look similar to Figure 5.1, where a study was conducted on why user testing with five users is enough and testing with more than five participants will add less and less new feedback for every extra participant.

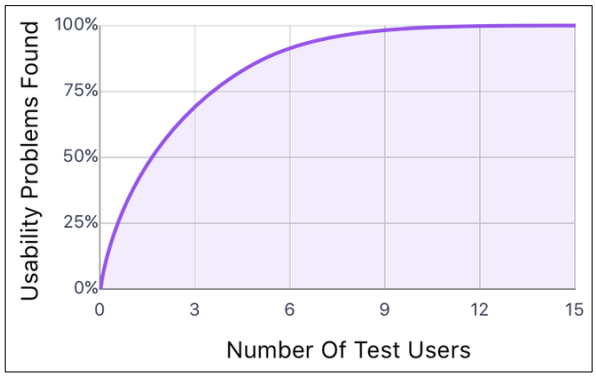


Figure 5.1: Usability problems found versus the number of participants.[2]

For this reason we aim to have five test subjects for this user study. The study will be conducted as a usability study, where the participants are comfortably seated in an office and someone of the team is taking notes on what they are doing. The comfortable environment should be created so that the user is able to make the same decisions as they normally would when they are not being 'tested'. This also means that this study will not take the shape of an interview, which means little to no interaction will take place between the participant and the team, who are purely there to observe.

5.2.2 Study

The user study will consist of a questionnaire, where the participant has to answer some short questions that assess the clarity of the visualisations used in the product, and is asked to complete some short tasks within the application. All questions will be setup according to the following three guidelines[3]:

[1] R. A. Burkhard. *Learning from architects: the difference between knowledge visualization and information visualization*. July 2004. DOI: 10.1109/IV.2004.1320194

[2] *Why You Only Need to Test with 5 Users*. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

[3] Clifford Chi. *The Beginner's Guide to Usability Testing*. URL: https://blog.hubspot.com/marketing/usability-testing

- Don't ask leading questions that insert your own bias or opinion into the participants' mind.
- Ask participants how they would complete a goal or task in the product or on the website instead of telling them what to do next.
- When participants ask "What will this do?," it's best to reply with the question "What do you expect it to do?" rather than telling them the answer.

The questions (except for question 1) were designed to investigate each feature of the product. Which feature is corresponding to which question is outlined in the table below, Table 5.1. The question form that was created and the form used for the team to write down the answers and observations can be found in Appendix C. As described before, notes were taken during the user study.

Table 5.1: Feature tested per question

Question	Feature
2	Searchbar
3	Info Button
4	Highlight
5	Checkbox
6	Focus
7	Filter
8	Compact

5.2.3 Methodology

The criteria for a successful visualisation are already defined in section 5.1. Then the answer to what extent the application is successful can be found by observing if the participants were able to identify known and unknown relationships between complex data during the user study. Furthermore, the results of the user study should provide us with a direct answer to sub-question two. However, the answer to sub-question three (what can be done to improve the visualisation) is also interesting, as the answer will include possible improvements to the application. We plan to analyse the feedback given by the users to improve our visualisation. More formally, we plan to convert the users' feedback into problem statements and given these problem statements, we would implement or propose a fitting solution. In order to prioritise the problem statements, we will look to categorise them in order to see which problem statements are most viable for improvement. We can define viable for improvement by answering the following questions on each user-study question:

Was the feature found? As a first constraint, it is important whether the participant is able to find the feature that was intended for each question. If they didn't, this issue should be addressed with high priority.

Was the feature used as intended? If the participant found the feature, but did not understand how to use it, the feature apparently needs to be easier to work with or needs some more explanation. When this happens, it should be solved in a second highest priority.

How long did it take to find the feature? When the first two categories are solved, it may be possible to improve the time it takes to find the feature. As it is very subjective when something takes to long, there will be no hard boundary on when this succeeds/fails.

How many clicks did it take to use the feature? The same as previous category, a subjective benchmark to indicate if there might be some features that take more mouse clicks than necessary.

5.2.4 Results

Company X set us up with seven participants to participate in the user study, both from the IT Department to the Legal Practice. However, one participant has been tested under different circumstances, as he had prior knowledge of the application. This participant has been left out of the result analysis. We will describe the rest of the results in order to find answers to the sub-questions "How does the user experience the current visualisation?" and "How can the result of the user study be used to improve the visualisation?".

User Experience

Most participants were excited and thought that the application really has potential if some additions or fixes take place that would take away some

of the difficulty in the usage. Also, two of the participants immediately saw opportunities to use the application for different kinds of data that is available at *Company X*. In the rest of this subsection we will analyse these problems and the next subsection will look into how this analysis can be used to find out the best way to improve the visualisation. This section will mostly mention the interesting results concerning answering the research question, the full results can be found in Appendix E.

Each user-study question underwent the four questions mentioned in subsection 5.2.3. As question 1 was a subjective question on the amount of information shown, this question was not included. The results are shown in Table 5.2.

Feature Found From the first column of Table 5.2, it is apparent that the users noticed the location of around 60% of the features. In a couple occurrences, where the participant missed a feature, the cause is how other features' presence overshadows the smaller ones, e.g. the location and size of the sidebar compels the user to focus on the right side of the screen and thus neglecting the searchbar located in the left side. Not only the features are being overshadowed, the graph visualisation itself was taken to the background. Often, after applying filters, the participants were still so fixated on the sidebar, that changes in the visualisation went unnoticed.

An observation that was made is that whenever participants missed the intended feature, they will take a lot of time to try and find the answer without using the feature or using the wrong feature. This is especially apparent in question 2, where the searchbar was the most efficient feature to use. If we take a look at the user study results in Appendix D, then we can see that the median of the time that it took for participants who used the searchbar was 39 seconds, while the median for participants who did not find the searchbar was 86.5 seconds to find the correct answer.

Usage Looking at the usage, only the participant that found the features were taken into account, as we can not say anything about the way they would be used by participants that did not find the features. When looking at whether the participants used the features as intended, we see some interesting results. The usage of the filter function in for instance question 7 is one of the questions that suffered from wrong usage. Only one of the three people that succeeded, used the filter in the intended way. When using the filter function in general (also for other questions), the observations was made that most of the participants used them the wrong way around. So, when a user wanted to see an employee with a certain interest, they filtered on the names of the interests instead of filtering out the employees that did not have that interest. By using the filters this way, it is still possible to find the correct answer however, it makes it slightly harder to instantly see the result in the graph.

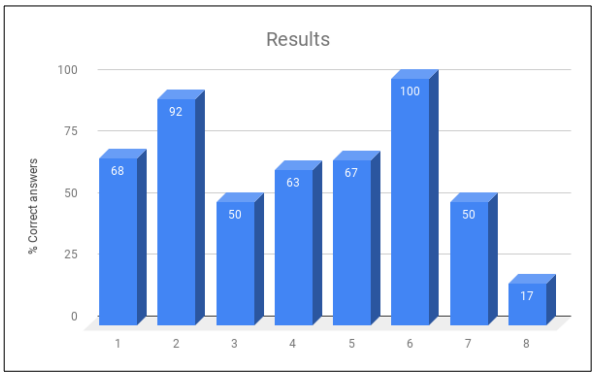


Figure 5.2: The percentage of correct answers in the user study (out of six participants).

If we combine the data from Figure 5.2 and Table 5.2, then we can obverse the percentage of correct answers where the corresponding feature was used as intended, if the feature was indeed found, of the corresponding question. What we see now is that as the session progresses, the percentage of intended use of features continues dropping. A couple explanations for this would be the following: participants simply did not find the correct feature or the correct feature proved too difficult to utilise. Another explanation could be the that participants are less likely to discover new features, as they try to stick to the features they already have discovered in previous questions.

Time The time each participant spent on each question was logged. Often a longer duration indicates that the feature was not found, where the

Table 5.2: Results of the user study (n=6).

	Feature Found	Used as Intended	Avg. Time (s)	Avg. # Clicks	Min. Clicks
Searchbar	4	4/4	54	2	2
Info Button	3	3/3	53	4,5	1
Highlight	4	3/4	119	7,75	5
Checkbox	4	4/4	144	10,5	2
Focus	5	5/5	56	5,7	2
Filter	3	1/3	189	22	14
Compact	1	1	264	20	6

participant will either make an educated guess, often with a doubtful substantiation, or just leave the answer blank. Another reason for a longer time was that if the feature was not used as intended, the participant would often use a different feature than the one intended and solve the question. For example, during question 5, participants would click on the node representing knowledge groups and count the employees in the list found in the sidebar, instead of visually inspecting the size of the knowledge group node in the graph to derive the answer. The answer is correct in the end, however a significant amount of time has passed.

Clicks In order to be precise and keep consistency, we only counted the mouse clicks that produced an actionable event e.g. clicking check boxes, clicking nodes or clicking the search bar. Mouse clicks on white spaces in the graph or miss-clicks on nodes were disregarded during the tally of the total.

The minimum number of clicks to complete each question is given in the table. We observe that all participants required more clicks than needed for all questions with exception of question 2. In case of questions 3 and 5, we can even see the average number of clicks needed by the participants are fivefold that of the actual needed clicks. Overall, the participants needed a little over two times the minimum number of clicks, most of these were made in order to find the right feature.

Comments During the user study, the participants made a lot of comments that were not necessarily linked to the question that they were executing at that moment. Some of the comments were repeated by multiple users, which indicates that they should be taken into account for future recommendations.

The following comments were deemed most important by the number of times they have been mentioned:

- Switching the context from explicit to compact is not fast enough.
- The filter function is not fast enough.
- The graph might move too much for some people (none of the participants themselves had issues with it, but they were thinking of others).
- Some of the nodes and their labels are too small to be clicked on or to be read.

Bugs During the User Studies, a few bugs came to light, which resulted in the need for a full reset (refreshing of the page) and should definitely be fixed. The following bugs were found:

- Filters do not reset correctly when a type is checked off and on.
- When checking all boxes off, the nodes of the last type remain visible.

How can the product be improved?

To answer the third sub-question, we need to identify the underlying problems of the user study results. Each participant gave valuable suggestions and wishes and while we appreciate them all, we have to keep the list concise. We have analysed and grouped the major issues.

Hard to find As mentioned earlier, more prominent features, e.g. the sidebar, visually overshadows the less noticeable features.

Several options are available to solve this issue. We could try to make the sidebar less prominent. Changes comparable with making the sidebar smaller or more transparent would contribute to this idea. Another possible solution would be to make the other elements, e.g. searchbar or context switch, more apparent on the screen. This can be achieved by putting the searchbar and the context switch in a similar box as the sidebar, as currently the searchbar blends in with the white background to some extent.

Not using the tool as intended Two different functions were not used as intended for different reasons. First of all, the filter function is apparently quite difficult to understand. As there are multiple options to use the filter and the application did not show any explanation about how to use these, they were used incorrectly. Because of this, an explanation has been added, as can be seen in Figure 5.3.

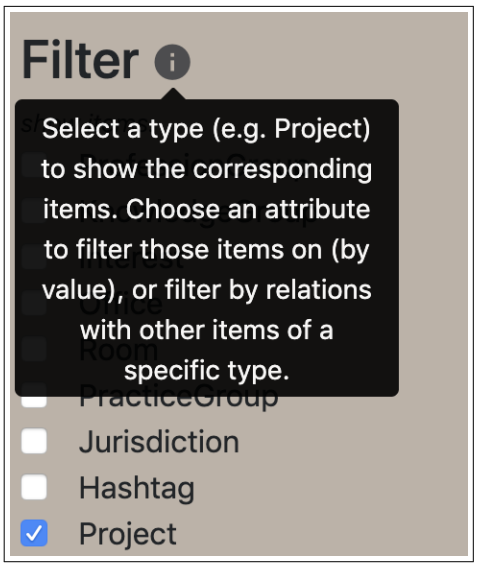


Figure 5.3: The filter function with explanation.

Besides that, some people did not use the implicit visual information about the node size in order to find out which one contained more relations to employees. Instead, they started counting the number of employees in the sidebar for each of the nodes. A possible solution for this can be a tooltip that shows the number of connections when you hover on a node.

Time We observed that users that took a relatively long time for a question, some of the tools were not used as intended, for which the solution has already been explained in the previous paragraph.

Another reason is about not finding the right tool, because the participants would use features that they had already used in previous questions. This issue would probably be solved over time, when the participant gains more experience in how to use the application. However, in order to speed this up, a tutorial on the use of this application could be very helpful.

Clicks The large deviation between the average mouse clicks and the minimum needed mouse clicks can be attributed to the unfamiliarity the participants have in the product. Experience gained from using the product in the long run, will steadily decrease the average number of clicks the users will do until they reach the minimum number of needed clicks.

Similarly to the solution to time described in the previous paragraph, a tutorial would also make a viable solution to reduce the extensive number of clicks. If users have already seen how to perform a certain task, then they will also know the most efficient way to perform that task. Perhaps a tutorial would be the optimal solution, as it would solve both the time issue and as well as the mouse click issue.

Suggestions In question 9, the participant was asked to make suggestions for additional features. The features that were mentioned most are:

- When a filter is applied or reset, the result of that query does not center, which made the graph go out of scope quite a few times. The suggestion made was to automatically center the graph on these queries.
- A lot of users mentioned they would have liked an intro at the first use, that would explain the most important tools of the application, so they had some basic idea of where to start.
- A reset button that would reset the entire graph to the begin state.

Comments The comments first two comments mentioned in Figure 5.2.4 involve the speed of the application. In the version of the application that was used during the user study, no indication was given when the application was loading. This caused the participants to click more often on the button, which made it even slower. Because of this, the Apply button of the filters will now be blurred, so it cannot be clicked until loading is finished, and will show a spinning loading graphic as well. Of course, the better solution would be to find out why the functions are this slow and to try to solve that.

Besides that, the comment was made multiple times that the graph might be moving too much for some people and that it would possibly cause nausea. As none of our participants showed any signs of nausea and some people rendered the movement unnecessary, while others found it 'cool', this movement is something that differs for each person and could be made into an on screen setting.

Finally, some nodes are too small to be clicked on, which also causes that their labels are too small to be read, which could possibly be fixed by making the labels clickable as well, which would make them easier to click on. Besides that the minimum size of a node and their label can be increased.

5.2.5 Conclusion

In this section, we will answer the sub-questions introduced in the beginning of this chapter. We defined the term successful in case of a visualisation. Based on the observations during the user study, we can say that the participants were indeed able to identify relationships between different data types. This is further supported by the fact that even though some participates were not able to find the intended feature, they were still able to solve the corresponding question.

Based on the results of the user study, the current user experience is quite satisfactory. This is not only evident from the number of correct answers given on the different user study questions, but also the positivity that the participants expressed about the product. However, each participant raised issues on their own as well.

We have analysed these issues and prioritised the problems statements which occurred the most number of times. Fitting solutions have been proposed in Figure 5.2.4 or implemented recently. Each solution improves the visualisation by making it easier for the user to understand certain features or create an overall better balance in the presence of different features. Now we can answer the main research question: ***"To what extent can the visualisations, i.e. explicit and compact, be intuitively grasped and provide the information they are supposed to provide?"***

By bringing the answers to the sub-questions together, we can definitely say that our visualisation succeeds to a certain extent, but needs quite some improvements in order to explore its full potential.

6 Organisation

This chapter will explain the organisation of the team to give an idea of how the project was carried out.

6.1 Division of labour

The product that had to be implemented could be split up into two main categories, being the front-end and the back-end. Keeping in mind that for both the front-end and the back-end an experienced team member was present, we decided to start with a (mostly) pair programming setup, where a team member with experience worked together with a member with less experience. After the proof of concept of the product was finished, it was possible to work more independently as everyone now had at least some experience with the used programming languages.

The writing of documentation in terms of reports (i.e. project plan, research report, final report) was something we all had to contribute to in order to ensure that all the available knowledge about the project would be transferred into the reports that were written.

6.2 Communication

Developing the right product requires good communication. We needed to maintain contact with three organisations: *Company X*, the coach and the team.

6.2.1 Client

In software development, it is important to define the project according to the company stakeholders' requirements and suggestions. For this reason we kept continuous and regular contact with *Company X* via several platforms, including WhatsApp, mail, Skype calls and face-to-face meetings. Additionally, our workplaces were in close proximity to the stakeholders, so emerging questions could often be asked and answered without much delay. At any time, we had a code branch ready that could be used to give a live demonstration of the product with its finished features (this is explained in detail in subsection 6.4.3).

6.2.2 Coach

To keep the coach in the loop, a weekly meeting was scheduled. During these meetings the progress of the project and the extent to which the educational interests of the TU Delft are served were discussed. The coach was kept up-to-date on our deliverables and provided iterative feedback via mail.

6.2.3 Team

In principle, the team met from Monday till Friday from 9:00am until 5:30pm to work on the project in the Office of *Company X*. This ensured that everyone put the necessary hours into the project and made it easier to work together on certain topics and start discussions when necessary. At the start of each workday we discussed which tasks we would take on that day and which tasks from the previous day we had trouble finishing, at the stand-up meetings.

At the beginning of the project, the team consisted of three members, being Toine, Marciano and Lex. However, during week 1 we received a message from Weilun, mentioning he was still looking for a group. After deliberation with the coach and *Company X* and a face-to-face meeting with Weilun, we were able to welcome him in our group that same week. While *Company X* arranged for Weilun's contract, NDA and access pass for the office, the rest of the team went to meet him at the TU Delft and got him up to speed.

6.3 Productivity tools

We have used a number of tools to work more efficiently. Below is a list that describes the most important tools.

Git

Git[1] is the *de facto* productivity tool in the software industry. Git offers several functions, including versioning, merging and branching of source code.

[1] `https://git-scm.com/`

GitLab

GitLab is an all encompassing development tool. Alongside its basic functionality as online repository for source code, it also offers, among other things: a CI/CD pipeline, an issues board and universal package management.

Trello

Trello[2] provides a neat overview of the tasks to be done, the tasks which are being worked on and the tasks which are already finished. We worked with Trello mostly at the start of the project and migrated to Azure DevOps in week 5, which provided similar functionalities.

Microsoft Azure

Microsoft Azure[3] is a cloud based computation solution for various applications. A feature we used of Microsoft Azure is DevOps, which provides continuous integration (CI), continuous delivery (CD) and continuous deployment, which is a combination of the former two.

Document Management System

DMS is used within *Company X* to work on documents collectively, securely and iteratively. The documents in DMS are stored on their servers and can only be accessed by employees which are permitted access. DMS stores which employee made an edit and earlier versions of documents are available for viewing. Since permissions to a shared DMS folder had to be configured first, we used Google Drive in the first week, and migrated existing files to DMS later.

Google Drive

Google Drive offers the online storage and versioning of files. It also provides the appropriate editing tools to make changes to the uploaded files.

Overleaf

Overleaf is an online LaTex editor. It provides online storage and back-ups of LaTex documents, as well as real-time collaboration and version control.

[2] `https://trello.com/`
[3] `https://azure.microsoft.com/`

6.4 Productivity tool usage

Productivity tools can yield a significant advantage if used thoughtfully. This is why we had a predetermined way of using these tools. This section will describe our processes regarding maintaining tasks, collaborating on documents, Git usage and Git hooks.

6.4.1 Maintaining tasks overview

We used Trello to add, order and distribute tasks among the team members. This way it was clear who worked on a task and which tasks remained unassigned. We invited one of the company stakeholders to view the Trello board, so they could view our progress, too. In week 5, we gained access to our own repository on Microsoft Azure, so we transferred our source code from GitLab to Azure and switched to the tasks board of Azure DevOps, which integrates well with Git. These tasks were evaluated at the daily stand-up meetings.

6.4.2 Collaborating on shared documents

Document sharing was an important part of information sharing between *Company X*, the coach and the team. For the research report and final report we, as team, used Overleaf, because we could edit the documents concurrently. We also created and used custom commands for leaving comments and suggestions on text written by other members of the team.

For less official documents amongst the team we used a shared folder on Google Drive. Google Drive also supports concurrent editing of documents, and has the benefit of not having to render the source text to view an up-to-date version of the document.

Sharing documents with *Company X* was done with DMS. DMS does not support concurrent editing, but does support versioning. So, when a document changed sufficiently, we exported the document to DMS with a new version number. Next, we informed the company stakeholders about the new

version of the document, so they could view it and provide feedback.

Documents meant for reviewing by the coach were mostly done via email, with the document exported as a PDF file. This made it easy for the coach to add feedback by placing comments at the relevant places in the document.

6.4.3 Git usage

In order to collectively work on the same code base we used Git. We have one main branch for the source code, `develop`, which should contain all features that were approved and tested by the team.

Each new feature or bugfix was developed on a separate branch with a descriptive name. When the feature was finished, the author of the accompanying code submitted a request to merge the code of this branch in the `develop` branch. Each member of the team was expected to inspect the added code and test the added functionality manually.

On big milestone updates (PoC, MVP and final product) a new version was released on the `master` branch.

6.4.4 Git hooks

Install packages

We added 'Git hooks', which automatically performs scripts when using Git commands. Each time a member switched branches `bundle install` and `yarn install` were called. These are the commands to install required packages using the package managers for Ruby and JavaScript libraries, respectively. This ensures that each member had the packages needed to run the application.

Generate documentation and check code quality

Each time code was committed on a branch, the documentation was updated using the documentation generators SDoc, for the back-end, and Vuese, for the front-end. These generators take comments that describe classes, methods, modules and functions and aggregate them into a searchable HTML document. Additionally, the code checkers RuboCop and ESLint, for Ruby code and JavaScript/*Vue.js* code, respectively, were run to reveal styling offences and bad coding practices. The committing of the code is obstructed if there are offences present that the code checkers could not automatically fix. This ensures up-to-date documentation and clean, maintainable code.

Run tests

When local code is pushed to the online repository, a *Git hook* is responsible for running automated front-end and back-end tests. Code could only be pushed when all tests succeeded, and together with the manual tests after every merge request, we could be reasonably certain that the added functionality did not introduce bugs or break the application.

7 | Conclusion

Company X asked for an interactive knowledge base of specific data that is scattered over multiple systems. This knowledge base should at least output the following:

1. which projects have been started around a certain topic,
2. who is involved in these projects,
3. what the status of these projects are, and
4. where these projects are carried out (e.g. which home market, which business unit).

After several meetings with *Company X*, a list of requirements was created. From this list of requirements, a PoC has been created in order to make sure the team and *Company X* were on the same page concerning the visualisation. When this was satisfactory, this PoC was used in order to create an MVP, which included all must have requirements except for the SP-link and AD, because there was a delay with the access to these properties. However, this version did include a visualisation of the data in a graph network, which was a should have and was also created with the knowledge that the application had to become generic in the future, which helped a great deal in the later stages of the development. With *Company X* being content with the MVP, a lot of ideas were proposed by *Company X* that would make the product more usable for them. When taking the scope of the project in consideration, these ideas were prioritised and implemented in that order, including the filter function.

This was also the time to start working on the implementation of the generic solution as explained in subsection 4.1.2. This way, the database and application are setup in a way to make as few assumptions about the contents as possible, while maintaining its usability. This has the benefit that this product can be used in different domains.

With respect to the development technologies, three main choices had to be made; the front-end, back-end and visualisation frameworks. Because no constraints were imposed by *Company X*, we made choices based on flexibility and ease of use, ultimately landing on the *Vue.js* front-end framework, *Ruby and Rails* for the back-end and *D3.js*, a popular data visualisation and manipulation library.

When the implementation phase of the project came to an end, it was time to find out ***"To what extent can the visualisations, i.e. explicit and compact, be intuitively grasped and provide the information they are supposed to provide?"***. To help answer this question, it was split up into three sub-questions, being:

- When is a visualisation successful?
- How does the user experience the current visualisation?
- How can the result of the user study be used to improve the visualisation?

Answering these sub-questions, by performing a user study (chapter 5) with seven employees of *Company X*, led to the conclusion that we can definitely say that our visualisation succeeds to a certain extent, but needs quite some improvements in order to explore its full potential.

As there was time left after the user study, some of the feedback that was given during the user study could already be implemented, while the rest of the improvements will be left for future implementation in section 8.1.

With regards to the original problem statement provided in section 2.1, the final product does bring the scattered data together in a dynamic, scalable and interactive fashion.

8 Discussion

8.1 Recommendations

Besides the recommendations made during the user study in Appendix C, there are some recommendations that the team wants to mention, which will be listed next.

1. As described in the section 3.3, a direct link between the product and existing data sources was not implemented. As a result, the product uses imported CSV files for its data. In order to automate this process an API endpoint should be implemented to receive the file, in the same way and with the same parameters as in the import page in the admin panel.
2. The filter function is very slow at the moment, especially when increasing the amount of data. Fixing this issue will lessen the frustrations felt of a unresponsive application by users and encourage them to use the application more. Currently, an indicator has been added to show the user that the application is still busy computing the data. However, this loading animation does not solve the underlying problem. A viable solution would be to cache data that has been computed before.
3. In line with previous recommendation, switching to the compact context takes increasingly long when increasing the amount of data. Unlike the previous recommendation, there is currently no indicator present to signal the user that the application is preparing to switch to the compact visualisation. Similar to the solution of the filter issue, a cache can be introduced to reduce the time needed to switch context.
4. Before deploying, there should be a way to protect the data from employees that are not permitted to view the data. As the Azure AD user permissions are not easily accessible, we were not able to adopt these permissions in our product. A workaround would be to shield the application with a login screen and introducing authenticatable user accounts with administrator or viewing permissions. While this is not hard to implement, the time needed to implement this has not yet been available, after it was known that we had to deviate from the original idea.
5. One suggestion that the *Company X* stakeholders made was to let users share networks and their configurations via URLs. While, in principle, each state in the front-end contains a separate URL that it sends as a request to the back-end to obtain the right data, these URLs are too long to be convenient. A function to encode and decode the information inside the URLs to allow for friendlier URLs would solve this problem.

8.2 Ethical implications

This section will elaborate on some of the ethical challenges that came with the nature of this project. Mostly, as the final product includes information on employees, privacy is a big concern. Also, when showing relations between employees, bias should be prevented as much as possible.

8.2.1 Privacy

The privacy of employees is an important ethical matter in the project. Currently, details about employees can be found inside the secure intranet of *Company X*. One could say that the information is already readily accessible and that the addition of the application inside *Company X* would not make a difference in the privacy of the employees. However, this is not entirely true. One could argue that the application makes it easier to access employees' details, i.e. it brings the employees of *Company X* more to the foreground. Furthermore, a matter that is not displayed on the intranet currently, are the projects in which employees are involved.

Currently, it is possible to adjust the details shown by the application with the aid of the admin panel described in subsection 4.1.1. Employees could indicate which attributes are shown to an admin. The removal of a few employee attributes, e.g. date of birth or personal addresses, would increase the privacy of employees and it would not impact the usability of the application. However, if employees decide to exclude a majority or all their projects, then the application would not be able give much relevant information.

8.2.2 Bias

The presence of bias in data can be discussed in various applications. Ranging from classification of skin cancer to stereotypes[1] in word embeddings.[2] Looking at the case of Esteva et al.,[3] the bias was introduced due to the deficiency of skin cancer of dark skinned people in the data. Thus the Stanford system were not able to detect skin cancer on dark skin as reliable as light skin.

The bias is in our case less explicit. If we combined the current graph network and the recommendation feature briefly mentioned in Appendix I, then the produced clusters of employees and projects could form a *filter bubble*.

One could be inclined to focus on only the same (or closely neighbouring) cluster of employees when filling new project teams. Then employees that are similar will most likely keep working with each other, which will decrease the opportunity for employees to work with colleagues they have not worked with and might lead to a sub-optimal composition of teams. Furthermore, working with the same group of people for an extended period of time proved to be a disadvantage, whereas a team of members with different talents performed better in problem solving.[4]

Thus, it may prove advantageous to aim to circumvent this filter bubble and promote more diversity in our application. One proposed solution is to place employees with interests or experience in different projects closer to each other.

[1] Andre Esteva et al. 'Dermatologist-level classification of skin cancer with ...' In: (Feb. 2017). URL: https://www.nature.com/articles/nature21056.

[2] Tolga Bolukbasi et al. 'Quantifying and Reducing Stereotypes in Word Embeddings'. In: (2016). URL: https://arxiv.org/pdf/1606.06121.pdf.

[3] Esteva et al., see n. 1.

[4] Steven R. Meier. *Building and Managing an Effective Project Team*. Sept. 2008. URL: http://connection.ebscohost.com/c/articles/34409947/building-managing-effective-project-team.

9 | Reflection

This chapter is dedicated to reflecting on the project. In the past quarter, several tasks went well, however, there is always room for improvement. We reflect on the process, the final product and the user study.

9.1 Process

In this section, the process around the project is reflected. The section is split into planning and contact.

9.1.1 Planning

At the beginning of the project, an extensive timetable was made, including all deadlines and planned milestones (including the deadlines for the different milestones (i.e. PoC, MVP etc.), split up in the different parts of the development process (i.e. front-end, back-end, visualisation, reports etc.). It was nice to have such an overview of when we had to work on what feature. Especially the milestones were well planned and were often ready a bit prior to the respective deadlines.

However, a lot of the documentation was sacrificed at first, taking liberties with the implementation. This was also noticeable for the final report. Despite the fact that an empty 'skeleton' version of the report was made a few weeks before the deadline, most content was written in the final few days.

9.1.2 Contact

Team

The contact within the team was quite a positive experience, which was most definitely supported by the fact that we had our personal working space at *Company X*. Interesting to reflect on within the team is the late addition of the fourth member. In hindsight, we think this process went quite well, including a face-to-face meeting with Weilun and a discussion with both *Company X* and the coach. This made sure everyone was on the same page before any promises were made. As we were still in the early stages of the development process, having just finished the project plan, the transition towards a four person team was relatively easy.

Client

The contact with *Company X* was also a positive experience, as they were always easily reachable via Skype (or just in the meeting room next door). One thing the team did need to adjust to, was the bureaucratic way some requests go within a big company like *Company X*. This was mostly noticeable in the time some requests took as they had to go through multiple layers within the company. A good example for this is the access to the real *Company X*-data.

Coach

As for the communication with the coach, the only noticeable thing to mention is the fact that everyone was busy, which sometimes made it hard to schedule face-to-face meetings. However, in the end no issues emerged from this as an empty spot in our schedules was found almost weekly and for the rest of the time, our coach was quick to reply to emails.

9.2 Product

The final product consists of almost all must- and should-have requirements as defined in section 2.3, except for the features linked to the access to real data and Azure AD. However, there are very useful functionalities that were not thought of during the construction of the requirements that were implemented to make up for those missed requirements. Overall, we are quite happy with how the requirements were achievable within the scope of the project.

When talking about an application like *Delphi*, there is no end in the possibilities for extra functionalities. This was noticed as in a lot of the meetings with *Company X*, new ideas were proposed in order to make the application even better. As at some point we did have to start thinking about finalising the product and starting on the final report, not all of the desired 'extra' functionalities have been implemented. However, the team has agreed with *Company X* that there will be some time after the project to finish up some of these functionalities to create that 'wow-factor'.

The way the requirements were setup, made the team highly aware of what the final product had to look like. The fact that the application had to become a generic and 'data content agnostic' application was already taken into account in the earlier stages of the development process, which made some of the changes to a generic product in the later stages a lot easier.

An issue we did find out too late, was how the application reacted to larger data sets. Mostly switching to the compact view and the filtering still take a lot of time to finish (20 seconds). For now, this is temporarily solved by giving the user feedback that it is loading. The better solution is of course to find out what slows it down, so we can improve on that.

9.3 User study

The user study went well overall. However, the product with which we conducted the user study ideally could have been better. Here we would not define 'better' as the future improvements that could have been made to the product, but as what the improvements of the features that were still in the pipeline would provide. There were a few instances during the user study where the participant would indicate an issue of which we knew it was already being implemented in parallel to the study.

Besides that, the duration of the individual user studies were twice as long as planned, a fifteen minute sessions became thirty minutes.

Bibliography

[1] Tolga Bolukbasi et al. 'Quantifying and Reducing Stereotypes in Word Embeddings'. In: (2016). URL: https://arxiv.org/pdf/1606.06121.pdf.

[2] R. A. Burkhard. *Learning from architects: the difference between knowledge visualization and information visualization*. July 2004. DOI: 10.1109/IV.2004.1320194.

[3] Clifford Chi. *The Beginner's Guide to Usability Testing*. URL: https://blog.hubspot.com/marketing/usability-testing.

[4] D. Clegg and R. Barker. *CASE Method Fast-track: A RAD Approach*. CASE method. Addison-Wesley Publishing Company, 1994. ISBN: 9780201624328.

[5] *D3.js - Performance Test*. URL: http://tommykrueger.com/projects/d3tests/performance-test.php (visited on 03/05/2019).

[6] Andre Esteva et al. 'Dermatologist-level classification of skin cancer with ...' In: (Feb. 2017). URL: https://www.nature.com/articles/nature21056.

[7] Steven R. Meier. *Building and Managing an Effective Project Team*. Sept. 2008. URL: http://connection.ebscohost.com/c/articles/34409947/building-managing-effective-project-team.

[8] Zach Nation. *3 Steps to Scalable Data Visualization in React.js & D3.js*. URL: https://www.codementor.io/znation/3-steps-to-scalable-data-visualization-in-react-js-d3-js-8t7kjxnk5 (visited on 03/05/2019).

[9] TU Delft. *BEPsys*. 2019. URL: https://bepsys.ewi.tudelft.nl/course_editions/7/projects/275 (visited on 24/05/2019).

[10] *What is SharePoint?* URL: https://support.office.com/en-ie/article/what-is-sharepoint-97b915e6-651b-43b2-827d-fb25777f446f.

[11] *Why You Only Need to Test with 5 Users*. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

List of Acronyms

A | Info Sheet

Project Delphi - *Interactive Knowledge Base*
Company: *Company X*
Presentation Date: *02 July 2019*

Description

Being a company with over 1500 employees, a lot of data is available about people and their day-to-day pursuits including projects they are working on. *Company X* has requested to gain more insight into this data, as it is currently scattered over multiple systems. Specifically, they requested to gain insight into who is involved in a topic, which projects have been started around this topic, what the status on these projects is and where these projects are carried out inside the company. The client wants this data represented in a dynamic, scalable and interactive visualisation.

To create a product that fulfils the expectations and needs of the client, a PoC was created. This basic version was used to make sure that the client and the project team had the same basic idea of the application created and was then extended to an MVP. Having completed most of this, as the access to the real data sources was delayed, additional features were implemented including a generic solution for the import of different data sets and a filter function.

The final product is a generic data visualisation tool, that makes it possible to visualise the data in a network visualisation. Manipulating this visualisation by filtering and searching, enables employees of *Company X* to gain more insight in that data.

In order to ensure that the visualisation can be used instinctively and deliver the information it is meant to, a user study was set up. In this user study, multiple employees of *Company X* tested the application by completing a set of small tasks. By tracking the different results (i.e. number of clicks, time, subjective opinions, usage of the features), the team was able to derive where potential improvements can be made. These improvements were then partially implemented and partially recommended for future implementation or research.

The most important recommendations for the future include improving the speed of the filter function and the context switch, but also making the connection to the *Company X* SP-pages, in order for *Company X* to actually use the product.

Members

Collective Contributions: setting up requirements, manual testing, writing reports, bug fixing.

Toine Hartman

Contributions: *Vue.js* component structure & logic, front-end testing, back-end testing.

Marciano Jorden

Contributions: database setup, data modeling, generalised solution back-end, API endpoints.

Weilun Chen

Contributions: user study, *Vue.js* components (front-end), D3 visualisation, front-end testing.

Lex Boleij

Contributions: user study, Searchbar and Sidebar components (front-end), D3 visualisation.

Client

Name: [redacted]
Affiliation: [redacted]
Company X
Name: [redacted]
Affiliation: [redacted] *Company X*
Name: [redacted]
Affiliation: [redacted]
Company X

TU Coach

Name: *O.E. Scharenborg*
Affiliation: *Intelligent Systems*

Contacts

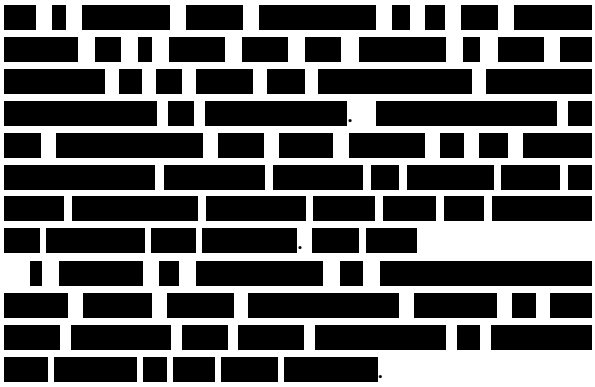
E: *Projectdelphi2019@gmail.com*

Repository

The final report for this project can be found at: `http://repository.tudelft.nl`

B | Project Description

Company Description



Project description (challenge)

We would like you to help us create an interactive knowledge base of specific data that is currently scattered over multiple systems.

We want to gain insight into:

1. who is involved in a topic,
2. which projects have been started around this topic,
3. what the status on these projects is, and
4. where these projects are carried out (e.g. which home market, which business unit).

There is a large amount of correspondence containing information about these projects, but the information is fragmented across different systems (e.g. our Document Management System, Active Directory, our Identity Management Solution or our SharePoint based corporate intranet). This project should make it possible to visualise the relations between these data in a dynamic, scalable and interactive fashion.

It will be up to you to decide on how the databases should be connected to each other and how you should get rid of similarities and duplicate information from the databases. A possible feature would be that the solution applies machine learning to make recommendations on issues such as the ones named above.

For example: We want to determine the best fit for future projects. Therefore we want to use information about employees such as their professional interests and which projects they have previously worked on.

The interactive knowledge base solution has to be accessible via browser and will be used by all *Company X* employees. You will also have to decide whether it is necessary to implement permission or role based governance for users.

What's in it for you

You will get experience working with the latest and greatest DevOps technologies such as Kubernetes, Microsoft Azure. We are always willing to discuss new technologies that you believe are right for the job.

Other information

We can offer you a workplace in our [redacted] or [redacted] office and we offer a competitive intern stipend (EUR 750,- per month on a full time basis) as well as compensation for travel expenses. As you may encounter highly confidential information during this project, you will have to sign a Non-Disclosure Agreement."[1]

[1] TU Delft. *BEPsys*. 2019. URL: `https://bepsys.ewi.tudelft.nl/course_editions/7/projects/275` (visited on 24/05/2019).

C User study: setup

Usability study

Welcome to the usability study for project Delphi and thank you for spending some of your time in order to make our product a success.

“We’re not testing you, we’re testing our designs, so nothing you do is wrong”.

We would like to ask you to think out loud as much as possible, so we are able to follow your thinking process.

0. Explore the application within 1 minute.

1. On a scale of 1 to 5, what do you think about the amount of information on your screen? (1 = too little, 5 = too much)

1	2	3	4	5

2. How many projects does employee Arlene Williamson work on?

3. And what is the MIC code of this employee?

4. Which projects do the employees of project Jacked America also work on?

5. Which two Knowledge Groups contain the least amount of employees?

6. Show only the employees that work on project Spilt.
   (**Only** this project and its employees are visible).

7. A project manager is looking for employees for a new project. Which employee would be most suitable to work on this project if you know that this project is interesting for employees with the following interests: *Articuno, Marowak, Shellder*

8. Which two projects have the most employees in common.

9. If you would have a magic wand, that could any feature to the application immediately, what would it be?

10. Do you have any questions or remarks about the product?

Usability study (answers)

2. How many projects does employee Arlene Williamson work on?

   3

   - *Klik op searchbar, zoek medewerker, lees af (sidebar / highlight)*
   - *minimaal 2 kliks (enter in de searchbar telt als klik): … kliks*
     - *Tijd : …*
     - *Gebruik gemaakt van searchbar?*
       - *Wel / Niet*
     - *Notes:*

       ……

       ……

       ……

3. And what is the MIC code of this employee?

   wila

   - *Klik op de URL naar info (1 klik) : … kliks*
     - *Tijd : …*
     - *Notes:*

       ……

       ……

       ……

4. Which projects do the employees of project Jacked America also work on?

   Winter Nuts, Pumpkin-spice Breaker, Jacked Star, Reg's Blend

   - *Selecteer project Jacked America, klik alle employees aan.*
   - *Kijk welke projecten worden gehighlight*
     - *Tijd : …*
     - *Notes:*

       ……

       ……

       ……

5. Which two Knowledge Groups contain the least amount of employees?

   Mining & Metals, Retail

   - *Voeg Knowledge Groups toe aan de graph (1 klik)*
   - *Kijk welke node het grootste is.*
     - *Tijd : …*

- *Notes:*

  ……

  ……

  ……

6. Show only the employees that work on project Spilt.
   (**Only** this project and its employees are visible).
   - *Filter toevoegen op Employee, Project kiezen, project The Town toevoegen.*
     - *Tijd : …*
     - *Notes:*

       ……

       ……

       ……

7. A project manager is looking for employees for a new project. Which employee would be most suitable to work on this project if you know that this project is interesting for employees with the following interests: *Articuno, Marowak, Shellder*
   - *Filter toevoegen op Employee, interest kiezen, 3 filters toevoegen.*
     - *Tijd : …*
     - *Notes:*

       ……

       ……

       ……

Sidney Morar

8. Which two projects have the most employees in common.

Pumpkin-pice Breaker	Winter Nuts

- *Sluit de URL van de employee (1 klik) : … kliks*
- *Ga naar de compacte view (2 kliks) : … kliks*
- *Dikte van de lijnen bekijken*
    - *Tijd : …*
    - *Gebruik gemaakt van compacte view?*
        - *Wel / Niet*
    - *Gebruik gemaakt van highlight?*
        - *Wel / Niet*
    - *Notes:*

      ……

      ……

      ……

D User study: results

Table D.1: The user study results per question in the questionnaire.

Question	Answer Found	Feature Found	Used as Intended	Time (s)	# Clicks
Searchbar	yes	yes	yes	37	2
	partially	yes	yes	22	2
	yes	yes	yes	40	2
	yes	no	-	74	5
	yes	no	-	99	11
	yes	yes	yes	39	2
Info Button	yes	yes	yes	62	4
	no	no	-	39	2
	no	no	-	26	1
	yes	yes	yes	68	4
	no	no	-	66	9
	yes	yes	yes	29	1
Highlight	yes	yes	yes	68	6
	yes	yes	yes	104	8
	no	no	-	89	8
	partially	yes	yes	144	7
	no	no	-	137	9
	yes	yes	no	160	9
Checkbox	yes	yes	yes	250	10
	yes	yes	yes	55	9
	no	no	-	43	5
	no	no	-	307	17
	yes	yes	yes	79	3
	yes	yes	yes	191	20
Focus	partially	yes	no	110	8
	partially	no	-	38	2
	partially	yes	yes	111	16
	partially	yes	no	28	2
	partially	yes	no	36	4
	partially	yes	no	13	2
Filter	yes	yes	no	214	26
	yes	yes	no	260	31
	yes	yes	yes	72	8
	no	no	-	257	12
	no	no	-	99	10
	no	no	-	203	15
Compact	no	no	-	126	10
	yes	yes	yes	261	20
	no	no	-	123	9
	no	no	-	127	3
	no	no	-	45	1
	no	no	-	225	12

E User study: comments

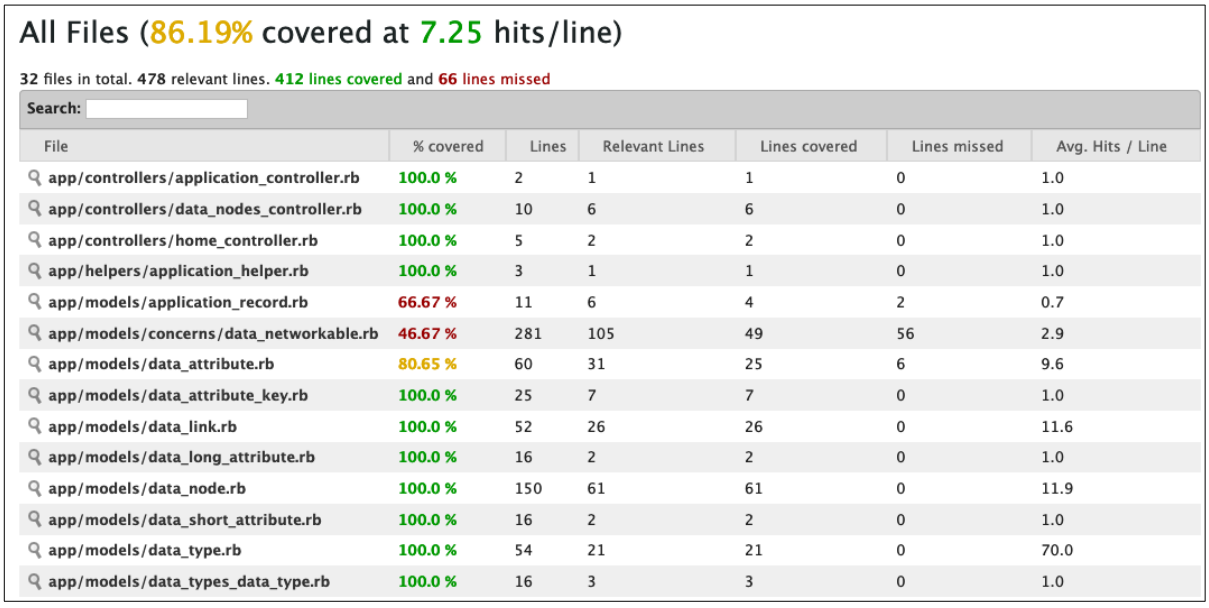
Table E.1: Comments that were made during the user studies.

Question	Comment	Endorsement
0		
1	Looks like a lot of info, but you need to know how to filter. Links and labels overlap sometimes, which makes them harder to read. As fake data was used, it does not mean much yet.	+1
2	When zoomed out too far, nothing is readable.	
3	You do not expect that you can click on 'info'	+2
4	Highlight path is not being used, instead the answer was taken per employee out of the sidebar	+3
5	The difference between node sizes are too close to each other.	
6		
7		
8	When nodes are in the middle of the graph, they probably have more mutual connections.	+1
9	Focus on multiple nodes. Make the contrast between colors bigger (colorblindness). Compact view: show a tooltip on hover of links. Be able to switch focus without losing your highlights. On focus: Enlarge neighbours to make them better readable. On filter: center the result. Add the support for multiple languages. Add a legend to explain what each node/color is. Make nodes of different shapes instead of just a different color. Get a tutorial on first use. Add a querytool. Collapsible option groups in searchbar A reset everything button	+3 +3 +2
General	Switching the contexts takes too long. Filtering takes too long. Does the graph move too much? Manually moving a node is quite hard as it returns to its original place. When zoomed out too much, the colors of the nodes are indistinguishable. When checking off all types, there are still nodes in view. When seeing different 'clusters' of nodes, they repel each other too much. Add highlighted path with ctrl/shift. Reset focus button is hard to find. Why not on escape or clicking behind the graph? I did not know you could zoom with the scroll button, maybe add a +/- button. The filter function is unclear (wrong way around). The difference between and/or filtering not clear.	+2 +2 +2 +1 +1 +2 +1 +1 +3 +2

F | Test coverage

```
============================== Coverage summary ==============================
Statements   : 83.53% ( 279/334 )
Branches     : 67.5% ( 81/120 )
Functions    : 83.77% ( 129/154 )
Lines        : 83.43% ( 277/332 )
================================================================================
-------------------------------------|----------|----------|----------|----------|-------------------
File                                 |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s
-------------------------------------|----------|----------|----------|----------|-------------------
All files                            |    83.53 |     67.5 |    83.77 |    83.43 |
 javascript                          |    81.18 |    62.69 |    74.67 |    81.08 |
  Network.vue                        |    80.56 |    60.32 |    73.97 |    80.45 |... 32,433,440,444
  utils.js                           |      100 |      100 |      100 |      100 |
 javascript/components               |    90.38 |    92.86 |    93.55 |    90.38 |
  AttributesTable.vue                |      100 |      100 |      100 |      100 |
  Buttonbar.vue                      |      100 |      100 |      100 |      100 |
  Searchbar.vue                      |    96.15 |      100 |    94.12 |    96.15 |               119
  Sidebar.vue                        |    81.82 |       90 |    90.91 |    81.82 |   144,156,157,158
 javascript/components/attribute_units |      100 |      100 |      100 |      100 |
  FieldAttribute.vue                 |      100 |      100 |      100 |      100 |
  ListAttribute.vue                  |      100 |      100 |      100 |      100 |
  TableRow.vue                       |      100 |      100 |      100 |      100 |
  UrlAttribute.vue                   |      100 |      100 |      100 |      100 |
 javascript/components/filter        |    82.35 |    64.86 |     90.7 |    82.14 |
  FilterBlock.vue                    |    92.86 |    83.33 |      100 |    92.86 |               110
  FilterGroup.vue                    |       72 |    42.86 |    83.33 |    71.43 |... 92,193,206,214
  FilterRow.vue                      |      100 |      100 |      100 |      100 |
-------------------------------------|----------|----------|----------|----------|-------------------
```

Figure F.1: Coverage of the front-end code base by Jest tests.



All Files (86.19% covered at 7.25 hits/line)

32 files in total. 478 relevant lines. 412 lines covered and 66 lines missed

Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
app/controllers/application_controller.rb	100.0 %	2	1	1	0	1.0
app/controllers/data_nodes_controller.rb	100.0 %	10	6	6	0	1.0
app/controllers/home_controller.rb	100.0 %	5	2	2	0	1.0
app/helpers/application_helper.rb	100.0 %	3	1	1	0	1.0
app/models/application_record.rb	66.67 %	11	6	4	2	0.7
app/models/concerns/data_networkable.rb	46.67 %	281	105	49	56	2.9
app/models/data_attribute.rb	80.65 %	60	31	25	6	9.6
app/models/data_attribute_key.rb	100.0 %	25	7	7	0	1.0
app/models/data_link.rb	100.0 %	52	26	26	0	11.6
app/models/data_long_attribute.rb	100.0 %	16	2	2	0	1.0
app/models/data_node.rb	100.0 %	150	61	61	0	11.9
app/models/data_short_attribute.rb	100.0 %	16	2	2	0	1.0
app/models/data_type.rb	100.0 %	54	21	21	0	70.0
app/models/data_types_data_type.rb	100.0 %	16	3	3	0	1.0

Figure F.2: Coverage of the back-end code base by Minitest tests.

G | SIG feedback: week 6

"Beste,

Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat een aantal aanbevelingen die meegenomen kunnen worden tijdens het vervolg van het project. Bij de evaluatie van de tweede upload kijken we in hoeverre de onderhoudbaarheid is verbeterd, en of de feedback is geaddresseerd. Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van hun code, en dient niet gebruikt te worden voor andere doeleinden.

Let tijdens het bekijken van de feedback op het volgende:

- Het is belangrijk om de feedback in de context van de huidige onderhoudbaarheid te zien. Als een project al relatief hoog scoort zijn de genoemde punten niet 'fout', maar aankopingspunten om een nog hogere score te behalen. In veel gevallen zullen dit marginale verbeteringen zijn, grote verbeteringen zijn immers moeilijk te behalen als de code al goed onderhoudbaar is.
- Voor de herkenning van testcode maken we gebruik van geautomatiseerde detectie. Dit werkt voor de gangbare technologieën en frameworks, maar het zou kunnen dat we jullie testcode hebben gemist. Laat het in dat geval vooral weten, maar we vragen hier ook om begrip dat het voor ons niet te doen is om voor elk groepje handmatig te kijken of er nog ergens testcode zit.
- Hetzelfde geldt voor libraries: als er voldaan wordt aan gangbare conventies worden die automatisch niet meegenomen tijdens de analyse, maar ook hier is het mogelijk dat we iets gemist hebben.

Mochten er nog vragen of opmerkingen zijn dan horen we dat graag.

Met vriendelijke groet, Dennis Bijlsma

Feedback

De code van het systeem scoort 4.1 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size en Duplication.

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.

Voorbeelden in jullie project:

- Employee.sidebar_data()
- Networkable.as_compact_links(included _association_names)

Bij Duplication wordt gekeken naar de hoeveelheid gedupliceerde code. We kijken hierbij ook naar de hoeveelheid redundantie, dus een duplicaat met tien kopieën zal voor de score sterker meetellen dan een duplicaat met twee kopieën. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om de hoeveelheid gedupliceerde code zo laag mogelijk te houden. Na verloop van tijd zal de gedupliceerde code moeten worden aangepast. Dit leidt niet alleen tot extra werk, aangezien op dat moment alle kopieën tegelijk moeten worden veranderd, maar is ook foutgevoelig omdat de kans bestaat dat één van de kopieën per ongeluk wordt vergeten.

Voorbeelden in jullie project:

- projectdelphi/app/models/employee.rb en
- projectdelphi/app/models/project.rb

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

Over het algemeen scoort de code dus bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase."

H | SIG feedback: week 9

"Beste,

Hierbij ontvang je de resultaten van de hermeting van de door jou opgestuurde code. In de hermeting hebben we met name gekeken naar of/hoe de aanbevelingen van de vorige evaluatie zijn geimplementeerd. Ook deze hermeting heeft het doel om studenten bewuster te maken van de onderhoudbaarheid van hun code en dient niet gebruikt te worden voor andere doeleinden.

Mochten er nog vragen of opmerkingen zijn dan hoor ik het graag.

Met vriendelijke groet, Dennis Bijlsma

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid is gestegen.

We zien dat de verbeterpunten uit de feedback op de eerste upload zijn aangepast, en op deze gebieden is dan ook een verbetering in de deelscores te zien.

Ook is het goed om te zien dat er naast nieuwe productiecode ook nieuwe testcode is geschreven.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject."

I | Research Report

DELFT UNIVERSITY OF TECHNOLOGY

COMPUTER SCIENCE
TI3806 BACHELOR END PROJECT

Project Delphi
Company

What is innovation?

Lex Boleij
Weilun Chen
Toine Hartman
Marciano Jorden

supervised by
Odette SCHARENBORG (TU Delft; Coach)
Otto VISSER (TU Delft; BEP Coordinator)
Huijuan WANG (TU Delft; BEP Coordinator)
[redacted] (*Company*)
[redacted] (*Company*)
[redacted] (*Company*)

3rd July 2019

Abstract

In the last phase of the bachelor's degree for Computer Science at the Delft University of Technology, students are required to successfully finish the *Bachelor End Project* (BEP). This paper will tackle Project Delphi, a Bachelor End Project hosted by *Company*. The client requested to gain insight into who is involved in a topic, which projects have been started around this topic, what the status on these projects is and where these projects are carried out inside the company. The client wants this data represented in an interactive visualisation. Alongside these requirements, we will encounter several challenges. As we are responsible for the entire product, we will split the development stack into three sections: front end, back end and visualisation. We will explore different frameworks and libraries for each development part and the challenges.

To ensure we work structurally and uphold certain deadlines, we created a timetable consisting of milestones. As students of TU Delft, we try to employ the standards of modern software development, including Git and Agile software development.

Contents

1 Introduction

This report presents the research plan of project Delphi. Project Delphi is a Bachelor End Project (BEP), hosted by *Company*. In the last phase of the bachelor's degree for Computer Science from the Delft University of Technology, students are required to successfully finish the BEP as a proof of competence. The project is supervised by [redacted] (company stakeholders) from *Company* and Odette Scharenborg (group coach) from the Delft University of Technology.

1.1 Company

[redacted]

1.2 Problem description

BEPsys description[1]: The client would like us to help them create an interactive knowledge base of specific data that is currently scattered over multiple systems.

The client wants to gain insight into:

1. who is involved in a topic,
2. which projects have been started around this topic,
3. what the status on these projects is and
4. where these projects are carried out (e.g. which home market, which business unit).

There is a large amount of correspondence containing information about these projects, but the information is fragmented across different systems (e.g. the Document Management System (DMS), Active Directory (AD), the Identity Management Solution or the SharePoint (SP) based corporate intranet). This project should make it possible to visualise the relations between these data in a dynamic, scalable and interactive fashion.

It will be up to us to decide on how the databases should be connected to each other and how we should get rid of similarities and duplicate information from the databases. A possible feature would be that the solution applies machine learning to make recommendations on issues such as the ones named above.

For example: We want to determine the best fit for future projects. Therefore, the client want to use information about employees such as their professional interests and which projects they have previously worked on.

The interactive knowledge based solution has to be accessible via browser and will be used by all *Company* employees. We will also have to decide whether it is necessary to implement permission or role based governance for users.

[1] Based on `https://bepsys.ewi.tudelft.nl/course_editions/7/projects/275`(visited on 25/04/2019)

2 Challenges

In this following section the most important challenges of project Delphi will be addressed.

2.1 Visualisation

As mentioned in chapter 1, a lot of data is scattered across multiple platforms at *Company*. It is interesting to see the relations between these data in a visualisation. This way it will be possible for employees to show this information to others. Some examples of interesting subjects to visualise will be explained in the following two subsections. Besides that, the client is working to make other data sources available in the future.

2.1.1 Projects

Questions like 'What is a certain employee currently working on?' and 'What are the employees within my department working on?' can be queried and answered on the spot with such a visualisation. Additionally, it would be possible to use the visualisation to make future decisions, e.g. to find the most suitable employees for a certain project to work on or to instantiate new projects based on projects that are absent.

The idea that came to mind during the first few meetings with the client, was a graph/network-like visualisation of which we created a basic demo (Figure 2.1) for the client.

Figure 2.1: Basic Demo Practice Group "Innovation"

It should be possible to navigate through such a network, which would give the user different content depending on what they are looking for. Consequently, the network has to be interactive and dynamic, based on the input of the user.

2.1.2 Case Laws

Another idea is to visualise which case laws resemble each other, and map their relations. A case law is a set of past rulings by tribunals that meet their respective jurisdictions' rules to be cited as precedent. Similar case laws might share similar articles and yield similar conclusions. This information can be reused to gain insight on current cases of this type. Some articles are used by thousands of case laws, and manually searching through them can be time consuming. Besides that, it is interesting to see metadata about these cases, including the location (website) of the the case laws found. Therefore, it is valuable to be able to intuitively find and filter related case laws in a visual network.

2.2 Source data

To give an overview on the different information platforms, this section will explain which data is available on the different subjects.

2.2.1 Projects

There is quite a lot of data available on the following subjects on the *Company* intranet about:

Employees. Basic employee information (*name*, *date of birth* etc.), work related information ([redacted]), personal interests, former projects.

Projects. A list of projects being worked on, containing information like *Project Owner*, *Project Members*, and other metadata.

Practice Groups. A Practice Group (PG) is a department within *Company*. Employees who belong to a PG work on a certain specialisation. Examples of practice groups are *Banking & Finance*, *Real Estate* and *Litigation & Risk Management*. An employee is

associated with a single PG. PGs generally operate on all disciplines (legal, notarial and tax) within their scope.

Knowledge Groups. A Knowledge Group (KG) is a (sub)market of clients, e.g. *Healthcare* or *Automotive*. It contains employees from multiple PGs.

Profession groups. A Profession Group expresses in which discipline an employee works. *Company* features the profession groups *Legal*, *Tax*, *Notarial* and *General* (also know as *Business Services*, i.e. *Finance*, *IT*, *HR* among others).

The challenge here is the fact that these data are scattered across multiple data platforms. The following platforms are some of the platforms that are most likely interesting for the product:

Inside is the intranet SharePoint (SP) page, accessible by all *Company* employees and houses company news, links to resources and the profiles of employees.

Knowhow. [redacted] is the SP page that contains Knowhow. Knowhow contains several lists, one of which is referred to within the company as '*The List*'. '*The List*' consists of projects that involve innovation within the company.

ICTweb (SP) contains a list of projects of [redacted] on the area of automated drafting, which is a project regarding innovation at *Company*.

Employeedata (*Odata-feed, JSON*) is an OData[1] Application Programming Interface (API) that serves stored data of employees in JavaScript Object Notation (JSON) format.

2.2.2 Case Laws

For the visualisation of the relations between case laws, there is a database called *Open Data*[2], which contains records of case laws, including the date,[3] the jurisdiction and the content/verdict. More interestingly, it shows incoming and outgoing relations to other case laws and articles of law.

[1] https://www.odata.org/

[2] https://www.rechtspraak.nl/Uitspraken/Paginas/Open-Data.aspx

2.3 Privacy & Sensitive data

The first information privacy law was already enacted in the early days of the Internet.[3] This shows how important the care of information data is. This should be reflected in our care of the sensitive data at *Company*. There are several directories present in the company. Like most big companies, some data in a department is not visible to other departments in the same company. We have to ensure in our visualisation of the data that employees belonging to one AD (A) should not be able to view data from another AD (B). Unless that employee of (A) receives an invitation to view such data from an employee belonging in AD (B).

2.4 Internet Explorer 11 support

The computers at *Company* are preconfigured with Internet Explorer 11 (IE11) as default browser and Google Chrome as optional. Although Google Chrome is more popular as a web browser in general,[4] the vast majority of the employees at *Company* still use IE11. We could categorise this preference as old habit, however, what is more important is the limitations that IE11 could impose on us during this project, especially on the visualisation aspect of the data. The proposed framework we prefer to use for the visualisation (D3.js, see section 6.3), uses the browser's native standards to create interactive representations of data.

Some implementation differences in these standards might cause the visualisations to behave differently in different browsers. Internet Explorer is infamous for having slightly different standards with regard to other major browsers. Although it is not completely transparent what components of D3.js are or are not supported by IE11, we can estimate the support for IE11 by comparing its compatibility with Scalable Vector Graphics (SVG), as SVG is used to display various types of graphics on the Web. Then we found that both web browsers

[3] A. A. Bushkin and S. I. Schaen. *The Privacy Act of 1974: a reference manual for compliance*. System Development Corp., 1976.

[4] Bipper Media. *Google Chrome Now More Popular Than IE, Becomes Most Popular Web Browser*. May 2012. URL: https://medium.com/bipper-media/google-chrome-now-more-popular-than-ie-becomes-most-popular-web-browser-8b676c9ae20a (visited on 26/04/2019).

support SVG, however it is possible that IE11 will not properly scale SVG files, i.e. certain visual elements, e.g. dots or lines, could appear smaller or larger than intended. Fortunately, these 'bugs' are well-known and suggested solutions often mention workarounds, including adding manual values for height, width and CSS fallback rules.

Internet Explorer 8 and earlier versions do not support HTML5, which is currently the latest standard for HyperText Markup Language (HTML). There are, however, libraries (i.e. HTML5shiv[5] and FlashCanvas[6]) that convert HTML5 elements into supported versions of these elements. We might be able to use this as an alternative method to render our visualisations if IE11 incorrectly renders them using the standard method.

2.5 Usage of libraries

A definition of 'reinventing the wheel' is to duplicate a basic method that has already been created or optimised by others. This notion also applies to software development in the following sense; When should an existing library be used to solve a problem and when is an own written implementation preferable? Although one could be tempted to search for a library every time he encounters a problem, there is merit in implementing your own solution. A good rule of thumb is to write your own code until you encounter something that is so common that there must be a ready-made solution for it. The usage of libraries does bring several benefits with it. Gouline notes the following;[7]

Reuse; The usage of libraries allows for the reuse of code by simply importing the libraries, instead of manually importing utilities from past projects. Although this is true, it will not play a big role in this project, as there is likely no old code to import.

Maintenance; Manually written code is only exposed to your application and its input. Hence, it is quite likely that you may missed some use cases and will only discover these cases if you somehow randomly stumble upon them. A well-maintained library has the benefit of feedback from dozens of developers, which will likely result in a quicker fix for such situations, provided that the library is still being supported.

While the aforementioned properties are mainly beneficial, one should not blindly follow libraries. Gouline also mentions the following points of caution while using libraries.

Some are good, some are bad; Not all libraries are good. It should be noted that every one can upload self-made libraries, which means that people who are not knowledgeable could provide a not optimal solution. On the other hand, a lot of them are also old and/or deprecated. A solution to this problem would be to manually inspect the code, or at least the methods that interests you. Otherwise, this could be the case in which writing your own implementation is more beneficial, instead of relying on a library.

Size; Some libraries are large and you feel disincentivized to import the whole library. This is especially true when you only need a few methods of that library. A solution to this problem would be to only import the methods that you need or to find a different, more compact, library to use.

Agreeing with Gouline, this project will mostly consist of own written code, unless there is a well-established library out there to use.

2.6 Responsive design

With web applications becoming more common in the last years, and devices becoming more diverse in terms of screen quality and size, the need for 'smart layout' applications arose.

Responsive design is a design paradigm which enables applications to adapt to various screen sizes. In our opinion this should be an implicit requirement for contemporary software, but as it came up in a discussion with the stakeholders, it is mentioned explicitly. Although mobile access does not have to be accounted for, the product will work on differently sized computer screens.

[5] `https://github.com/aFarkas/html5shiv`

[6] `https://github.com/everlaat/flashcanvas`

[7] Mike Gouline. *Library-driven development*. Sept. 2015. URL: `https://blog.gouline.net/library-driven-development-992b561eb5f1` (visited on 26/04/2019).

2.7 Generalised solution

In order to have the product integrate data from new sources, generalised import functionality is needed. Such a solution makes little assumptions about the structure and attribute names of the data and can handle multiple data formats. The solution should be relatively easy to use without requiring too much knowledge about the internal components of the product. An idea is to have the user decide which attributes in the new data source correspond to which attributes that can be viewed on the screen. Not knowing the data beforehand means that displaying the data comprehensibly can be challenging. This is because tweaking the visualisation after the addition of new data is impossible.

2.8 Conflict resolution

The data sources need not have disjoint data. This means that some records i.e. product or employees may appear multiple times across the data sources. These records might be slightly different because of out-of-date information, human error or punctuation differences. The information needed to create a solution that automatically uses the right information every time, will most likely not be available. However, a function can be implemented that alerts users when there are multiple records that need to be merged and let them choose which information is correct.

2.9 CRM integration

A good addition to the visualisation of the data is Client Relation Management (CRM) data integration i.e. add useful data from clients to the display in the product. However, as is already mentioned in 2.3, data is quite sensitive. Thus it is unsure whether client data will be available to implement such a function.

2.10 Employee recommendations

Since the product will be used to make better decisions about which employees are the best fit for new projects, it could be interesting to integrate employee recommendations into the product. Information on what data is relevant for assigning employees to projects and whether this information exists in at least one of the data sources has to be available. Additionally, the weight of importance for this data has to be available.

2.11 DMS integration

It is already known that the existing DMS will be replaced in the near future. The new system might be very different from the existing one. Then it would be illogical to include this in our product. However, if our product is able pass the requirement mentioned in 2.7, then our product would be able to process the substitution of DMS.

3 Research

Visuals has always been a strong tool in aiding people in communicating and understanding. The degree of difficulty of visuals can range from simple advertisements to complex concepts in various academic fields. The visualisation of data has become an increasingly popular and powerful tool in recent years.[1] However, Few notes several bad trends of visualisation. One of which he describes how current visualisation applications focus largely on flashy aesthetics instead of functionality. The visualisation of the data is a big part of this project. Choosing the right tools and methods are essential to a good product. In the next section, we will try to establish which visual tools proof to be beneficial to our product.

3.1 Visual tools

As mentioned in 2.1.1, we adopted a graph/network-like visualisation early on. A graph proofs to be well-suited for this project, as a graph is able to show the presence or absence of relationships between objects or sets of objects.[2] Although the graph shown in Figure 2.1 is quite simple, we could make it as complex as the client desires.

3.1.1 Existing tools

More formally, Figure 2.1 describes only entities within *Company* as nodes and relationships between these entities with edges.

One could extend the graph with weights on the edges. The weight represents the similarity of the two connected nodes. Then this graph would produce a different visualisation, i.e. nodes with an edge of smaller weight are drawn further apart while edges of larger weight are drawn closer together. The resulting visualisation would produce *cluster* of nodes in different areas of the graph. Users would be able to identity which nodes are similar at an initial glance. These *weighted* graphs would be beneficial for the client, as they would be able to see which projects are of similar nature as result of the employees connected to them.

Another promising model is the so-called *Force-directed graph*.[3] Unlike weighted graphs, where the position of the nodes are determined by the weights of the edges, the nodes of the force-directed graph are initialised in random positions. Then, the algorithm applies some kind of force to these nodes to determine their final position. However, the algorithm will try to keep neighbouring nodes closer together than unrelated nodes. This property is favourable for our visualisation as we do not have to specify the distance, in this case the weights of the edges, of our nodes. We could increase the repulsion of unrelated nodes to further increase the distance when they are drawn, thus creating an even more structured graph.

3.1.2 Our implementation

The client wants to visualise various elements which should be view-able from different aspects, i.e. the client wants an interactive graph which presents the various sets of data optimally depending on said data. Then we find it hard to stick with one graph implementation. However, the *Force-directed graph* provides a starting point. Then we could extend on this model by introducing different elements based on the clients' wishes.

We propose a user experience study to collect structured feedback from the clients and to process this feedback into the product. To further emphasise this work flow, we have split up the end product in iterative milestones described in chapter 5.

3.2 User study

The proposed user study will ensure that we find the right visualisation for the client. The user study will be done in the course of the project. The method we will use can be described as follows:

Procedure; We will ask the client to evaluate our application during demonstrations/showing.

[1] Stephen Few. 'Data visualization pas, present and future'. In: (2007). URL: https://www.perceptualedge.com/articles/Whitepapers/Data_Visualization.pdf.

[2] Chun-houh Chen, Wolfgang Härdle and Antony Unwin. *Handbook of Data Visualization*. Springer, 2016.

[3] Michael J. Mcguffin. 'Simple algorithms for network visualization: A tutorial'. In: *Tsinghua Science and Technology* 17.4 (Aug. 2012), pp. 383–398. DOI: 10.1109/tst.2012.6297585. URL: https://ieeexplore.ieee.org/document/6297585.

Participants; The stakeholders will only consist of us and the clients, as this project requires a Non-Disclosure Agreement (NDA). However, there is a possibility to extend the participants with other employees within *Company*.

Measures; We will construct an assessment form for the client to fill in after a demonstrations/showing. A written assessment form will provide structure to this user study.

Analyses; Based on the evaluation of the clients, we can decide which feedback to implement and which existing feature to change.

Although the feedback from the clients is important, we, as developers, can also provide suggestions, e.g. combining aspects from different visualisations.

3.3 Ethics: Bias

When introducing recommendations and visualisations the ethical aspects of its consequenses should not be ignored. One aspect that comes to mind during this project is bias. The presence of bias in data can be discussed in various applications. Ranging from classification of skin cancer to stereotypes[4] in word embeddings.[5] Looking at the case of Esteva et al., the bias was introduced due to the deficiency of skin cancer of dark skinned people in the data. Thus the Stanford system were not able to detect skin cancer on dark skin as reliable as light skin.

The bias is in our case less explicit. If we were to introduce the graph network mentioned in 3.1 and the recommendation feature briefly mentioned in 2.1.1, then the produced clusters of employees and projects could form a *filter bubble*. Although the filtering (of potential interested employees) is not a result by an algorithm/AI,[6] the reduced diversity in the amount of recommendable employees can still be categorised as a filter bubble. Here, we will use the term diversity, not necessarily in the sense of skin colour or cultural background, but more in the sense of informational and personal attributes, e.g. talents, project background and personality. Although skin colour and cultural background is important, our application will not have direct insight in such information.

One could be inclined to focus on only the same (or closely neighbouring) cluster of employees when filling new project teams. Then employees that are similar will most likely keep working with each other, which will decrease the opportunity for employees to meet new colleagues. Furthermore, working with the same group of people for an extended period of time proved to be a disadvantage, whereas a team of members with different talents performed better in problem solving.[7]

Thus, we should try to aim to break this filter bubble and promote more diversity in our application. One proposed solution is to place employees with interests in projects where he/she has no connection to closer to each other.

[4] Andre Esteva et al. 'Dermatologist-level classification of skin cancer with ...' In: (Feb. 2017). URL: https://www.nature.com/articles/nature21056.

[5] Tolga Bolukbasi et al. 'Quantifying and Reducing Stereotypes in Word Embeddings'. In: (2016). URL: https://arxiv.org/pdf/1606.06121.pdf.

[6] Judith Moeller and Natali Helberger. 'Beyond the filter bubble: concepts, myths, evidence and issues for future debates'. In: (June 2018). URL: https://www.ivir.nl/publicaties/download/Beyond_the_filter_bubble__concepts_myths_evidence_and_issues_for_future_debates.pdf.

[7] Steven R. Meier. *Building and Managing an Effective Project Team*. Sept. 2008. URL: http://connection.ebscohost.com/c/articles/34409947/building-managing-effective-project-team.

4 Requirements

In order to build the right product for the described problem a set of requirements was compiled. These can either be classified as *functional* or *technical*.

Functional requirements specify needs that are visible to the end user, e.g. capabilities of the product or specifics on the layout. *Technical requirements* specify 'invisible' constraints, e.g. requirements on the development process or code quality.

The requirements are prioritised conform the *MoSCoW*[1] method. This method is used to separate the requirements into *must-haves*, *should-haves*, *could-haves* and *won't-haves*.

Must-haves are requirements critical for developing the right product. Without them the product would not be right for the described problem. **Should-haves** are less important than *must-haves*, but still add a lot of value to the product. **Could-haves** have less importance than *should-haves* and have a lower value to development time ratio or have dependencies that are uncertain at this moment. **Won't-haves** will not be implemented, because they do not fit in the scope of this BEP.

4.1 Functional requirements

Must have

Link data from existing data sources. Without this link we can not have an up-to-date representation of the company data.

Overview of projects with statuses. The main purpose of the product is to aggregate and show information about existing projects and its employees.

Overview of people in projects with interests and expertises. The main purpose of the product is to aggregate and show information about existing projects and its employees.

Show projects that involve innovation. The company explained that they intend to use this product to show what the company is doing with regard to innovation.

Detect and alert about conflicting data from different sources. Aggregating data might yield duplicate records, and these records might not be identical if one of the data sources is not kept up-to-date (or if there are typos). By detecting conflicting data, steps can be made to make the shown data more accurate.

Accessible from the company's intranet. The end product is intended to be used by the companies' employees. The intranet shields from access from outside the company.

Accessible with *Azure Active Directory* Single Sign On (SSO). Company employees already actively use an *Azure* AD account, so this integrates well with the company software infrastructure. Having to create a separate account would discourage employees from using the product.

User permissions. Not everyone should have access to all data. Some data is not meant to be seen by everyone.

Should have

A visual graph linking projects to users. To create a clear overview of the information, a visual layout is an effective and intuitive solution. An overview is possible in a static HTML format, but lacks the interactive elements. Therefore, a hands-on interactive visualisation should be integrated. Depending on the element (e.g. a project or a user) that has the focus relevant information and links to neighbouring elements should be displayed. Irrelevant information should be hidden to keep the layout comprehensible.

Generic solution for adding new unspecified data sources. The company has requested that they might want to add new data sources in the future. As we cannot know the specific structure of these sources in advance a generic solution would be optimal. The solution should accept new data

[1] D. Clegg and R. Barker. *CASE Method Fast-track: A RAD Approach*. CASE method. Addison-Wesley Publishing Company, 1994. ISBN: 9780201624328.

about or linked to projects and employees, having a SQL or CSV format.

User permissions via *Azure Active Directory Groups*. Synchronising user permissions with *Active Directory Groups* permissions can eliminate the need to manually keep the user permissions within the product up-to-date.

Editable user permissions. Users with certain (administrative) permissions should be able to edit permissions of other users.

Could have

Employee recommendations for projects. An interesting extension (proposed by the stakeholders) is to incorporate a recommender system. This could, for example, suggest certain employees to work on a new project, based on their interests or past experience.

Integrate client data. Integrating client data adds useful extra features to display in the product, but since this information is confidential it is unsure whether we can acquire permission to view and process this data.

Virtual Reality traversal of visual graph. The use of Virtual Reality (VR) is a nice-to-have. However, the company does not provide VR-devices for their employees by default. The implementation of VR requires a lot of effort while only yielding a 'gimmick'. Because of this, it is classified as a *could-have*.

Won't have

Data source: Document Management System. Although the stakeholder proposed the DMS as a data source, there are plans to replace this system in the near future. The new system is not likely to be similar to the old one, which is why it would be illogical to include this in our product. However, if we meet the requirement of having a generic solution (see page 8), the new system can be added afterwards.

4.2 Technical requirements

Must have

Clear, up-to-date documentation It is important that *Company* is able to continue development of the product after the BEP. Therefore properly documented code is a must.

Should have

Support for Internet Explorer 11. The majority of *Company* employees use IE11 by default, but all employees have access to the *Google Chrome* browser. Having to develop for a browser which lacks support for many Cascaded Style Sheets (CSS) and JavaScript functionality might cause the need to either create workarounds or compromise the quality of the end product.

An automated code checker for code formatting. This encourages to write code according to a defined and well-structured format, which helps in future maintainability.

Easy-to-maintain codebase. A low code coupling and more generic methods can reduce bugs and allow both us during the BEP project and the company after the handover to add or change functionality more easily.

Unit tests. To gain confidence in the correct functioning of the product, each unit should be tested to confirm the behaviour is as expected.

Integration tests. To gain confidence in the correct functioning of the product, the units should be tested together to confirm that their combined functions are working as expected.

Continuous integration for automated testing. A continuous integration pipeline allows for efficient regression testing.

A minimal code coverage in tests of 80%. A higher code coverage[2] ensures that the tests cover a greater portion of the code of the product. However, this does not say anything about the quality of the tests. Making sure each test we have covers

the entire functionality, seems more lucrative than making sure everything is tested. Especially as for some code it costs a lot of time to create the tests and the resulting tests would be highly complex. When there is a change in functionality of the code, it would really be a burden to maintain these complex tests. For these reasons we have decided that 80% is reasonable because this does ensure that a satisfactory amount of the code is covered by tests, but more importantly, the quality of the tests will be better as well.

Could have

Application: responsive design. Multi-device support is a nice-to-have, but we expect that the majority of users will interact with the product on a desktop computer or laptop. Therefore, responsive design does not yield a significant benefit. (See section 2.6).

Add or edit data via the application. The systems providing the existing data can be used to add or edit data if needed. However, if possible, it would be useful to be able to modify data from the application.

Won't have

We do not have any technical won't-have requirements.

[2] Code coverage is a metric to describe to which degree a code base is tested. The testing framework automatically determines whether each line of code has been run in tests and calculates the total coverage (as a percentage of the total amount of lines of code).

5 Milestones

This section will explain the milestones that will be used as a guideline throughout the project. A general time line that was created at the start of the project is shown in Figure 5.1 below, where 'W3' stands for Week 3 of the project.

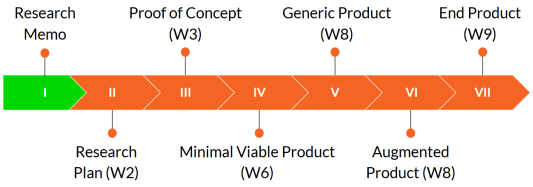


Figure 5.1: Milestones Project Delphi

5.1 Research Memo

The research memo is a brief document that describes the problem that must be solved, along with a proposed solution to address this problem. This document incorporates the requirements that *Company* has for the solution. The research memo will be used to sketch a general idea about the end product in an early stage in order to see whether this aligns with the companies' idea of the end product. Due to the feedback that the company or the coach will provide, the end product may differ from the information of this document.

5.2 Research Plan

The research plan (this document) is a more detailed version of the research memo. It contains a detailed description of the proposed product and its components and justifies the design choices. It takes into account the feedback of the company and coach and will be adjusted accordingly. The end product is not expected to differ a lot from the description in this document.

5.3 Proof of Concept

The Proof of Concept (PoC) is a basic version of the product. It serves to illustrate the idea and should demonstrate whether the design of the desired solution is suitable for the intended goal. This version may not be fully operational.

5.4 Minimal Viable Product

The Minimal Viable Product (MVP) is the bare minimum product that would solve the project's main problem. The MVP implements all *must-have* requirements. It is used to make early testing possible, as well as to make the product tangible, so more detailed feedback can be given and taken into account.

5.5 Generic product

Contrary to the MVP, the generic product will have *should-have* requirements added to the product. Examples of these additions can be implementing a dynamic, scalable and interactive visualisation of the data, or adding a generalised way to handle importing data from data sources. These additions will add to the value the product will have to the client.

5.6 Augmented product

The augmented product will have additional features that improve the product above expectations. This corresponds with the *could-have* requirements. These have a lower priority and will only be implemented if there is enough time. An example of such an augmentation would be the support of a VR visualisation or machine learning capabilities.

5.7 End Product

The end product will be the final product that will be handed over to the client. This product will have only finished and working features, including clear documentation on how to maintain and/or extend the product in the future.

6 Frameworks & technologies

In this section, we discuss the choice for several different frameworks and libraries which will influence the development process. The discussion will be split over the following sections: Front end, Back end and Visualisation.

6.1 Front end

The front end is the part of the application that handles the display of all different elements and the navigation through them. We considered several frameworks:

AngularJS[1] is quite a complex framework with a steep learning curve. Furthermore, it has strong opinions on how to structure the application, instead of being modular and flexible. Currently, AngularJS is still supporting IE11, however there might be a few points that require attention. Most of them are syntax related issues and their respective fixes are clearly documented[2].

Angular[3] is the successor of AngularJS. This is the framework ranks highest in popularity according to CodeAnywhere[4] and it performs well on speed performance benchmarks.[5] However, this framework is complex as it mainly targets large applications. It also practically requires the knowledge of TypeScript[6,7], as this is the language used in documentation and examples. Similar to AngularJS, Angular is compatible with IE11, with the support from polyfill scripts[8] if necessary. Concluding, Angular has a steep learning curve with support for IE11. However, we apprehend that the learning curve will slow down our quick development iterations.

Ember.js[9] is powerful but it uses a lot of conventions. Therefore is it hard to learn and we think we do not have the time to wrap our head around such an extensive framework in the given time. Besides, speed performance benchmarks show that some operations can take three times as long compared to Angular and Vue.[10]

Target builds of Ember will not include IE11 on default. However, it is still possible to configure IE11 compatibility.

React[11] is a capable yet flexible framework that shares a lot of similarities with the Vue framework. Similar to the prior frameworks, React supports IE11 with the aid of global polyfill scripts.[12]

Vue[13] is very similar to React. It is both powerful and versatile. One reason we prefer this over React, is that one of the team members already has experience with the framework. Besides, it is easy to learn[14] so it should get us started quickly. Speed performance benchmarks reveal that Vue is only slightly slower than Angular.[15] This is not likely to throttle the application performance, because the bottleneck will lie in data acquisition and processing.

Additionally, Vue offers the ability to use *Modern Mode*[16], which means that Vue offers the possibility to reduce the amount of heavy and less efficient code that is needed to support older browsers, because most current browsers already have in-built support for these functionalities. *Modern Mode* creates two versions of the application:

[1] https://angularjs.org/
[2] https://docs.angularjs.org/guide/ie
[3] https://angular.io/
[4] https://blog.codeanywhere.com/top-10-most-used-javascript-frameworks/
[5] S. Alexander. 'Speed Performance Comparison of JavaScript MVC Frameworks (Dissertation).' Bachelor thesis. Blekinge Institute of Technology, 2015. URL: http://www.diva-portal.org/smash/get/diva2:998701/FULLTEXT03.
[6] https://www.typescriptlang.org/
[7] Eric Wohlgethan. 'Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js'. Bachelor thesis. Hochschule für Angewandte Wissenschaften Hamburg, 2018.
[8] https://angular.io/guide/browser-support#polyfill-libs
[9] https://emberjs.com/
[10] Alexander, see n. 5.
[11] https://reactjs.org/
[12] *JavaScript Environment Requirements – React*. URL: https://reactjs.org/docs/javascript-environment-requirements.html (visited on 02/05/2019).
[13] https://vuejs.org/
[14] Wohlgethan, see n. 7.
[15] Alexander, see n. 5.
[16] https://cli.vuejs.org/guide/browser-compatibility.html#modern-mode

1. A reduced bundle for modern web browsers, e.g. *Google Chrome, Firefox or Microsoft Edge*.
2. The standard bundle, which includes support for legacy browsers.

We prefer the Vue framework for its flexibility and comprehensibility. Therefore, we will use Vue in our application, although other options are not necessarily inferior.

6.2 Back end

The back end is responsible for sourcing and aggregating all data and delivering it to the front end. It also handles authentication/authorisation. We compared the following frameworks:

Django[17] is a Python based back-end framework. It is widely used framework as the Python language offers elegant yet concise lines of code. However, similar to React, we don't have any experience with it.

Flask[18] is, similar to Django, also a Python based back-end framework. A key difference is fact that Flask is a so-called microframework, meaning that Flask does not require tools or libraries and it has no database abstraction layer. As this is the first time for us to encounter microframeworks, we think it would be wise to stick to what we know.

Express[19] is a Node.js based framework. It is particularly well versed in applications that have all their states in memory. It could then provide responses more quickly to requests by eliminating the reading/writing of states into a database. A drawback is the steep learning curve[20] and the fact that Node.js works with registered callbacks. This callback nature is proven to be a bit difficult to understand for beginners, resulting in the so-called 'callback hell'[21].

Ruby on Rails[22] is a Model-View-Controller (MVC)[23] based web framework in Ruby. Ruby is a modern interpreted programming language which values convention over configuration. This ideology was adopted by Rails. Each Rails project has a predefined folder structure and function parameters are set to default values unless explicitly changed. This results in readable, concise code with low repetition. It is used by tech companies as Twitter[24], AirBNB[25], and Github[26]. Additionally, one of the team members is experienced with Rails, which helps to get the rest of the team started quickly.

Thus, we opted for Ruby on Rails, as we prefer the robust and convention valued programming aspects of the framework.

6.3 Visualisation

The visualisation is arguably the most important part of the development stack, as this is a big part of the front-end display that users will see and interact with. We compared the following frameworks and libraries:

D3.js[27] is a JavaScript based library. It's main purpose is the manipulation of data into interactive visualisation using HTML, SVG and CSS. D3.js achieves this by allowing you to bind arbitrary data into the Document Object Model (DOM), and then transforms the object by data-driven transformation.

D3 is capable of handling large amounts of data. Visualising everything at once will, however, reduce responsive of the browser. Re-

[17] https://djangoproject.com/
[18] https://palletsprojects.com/p/flask/
[19] https://expressjs.com/
[20] Tyler Crawford and Tauqeer Hussain. 'A Comparison of Server Side Scripting Technologies'. In: *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2017, pp. 69–76.
[21] http://callbackhell.com/
[22] https://rubyonrails.org/
[23] Glenn E. Krasner and Stephen T. Pope. 'A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80'. In: *J. Object Oriented Program.* 1.3 (Aug. 1988), pp. 26–49. ISSN: 0896-8438. URL: http://dl.acm.org/citation.cfm?id=50757.50759.
[24] https://twitter.com/
[25] https://airbnb.com/
[26] https://github.com/
[27] https://d3js.org/

ports on D3's visualisation performance differ[28,29]. However, for a visualisation to be clear and legible, it should not contain too many elements (in the order of hundreds). Therefore a solid dynamic (depending on context and number of selected data points) server-side abstraction is essential.

JavaScript InfoVis Toolkit[30] is a JavaScript based toolkit for creating several visualisation figures, like hyperbolic and space trees. Although there is no information on the website about the compatibility with IE11, the creator of the framework announced[31] that all major web browsers are supported. Unfortunately, no examples of using *JIT* with large volumes of data can be found and has not been maitained since late 2014. This does definitely not encourage to use the library.

ZoomCharts[32] is a rich library of charts. The charts are visually pleasing and are compatible with IE11[33]. The downside, however, is that a paid licence is required to use this library.

Cytoscape.js[34] should not be confused with Cytoscape. Although the two are similar in design concepts, they are completely independent from another. Both are primarily used for the visualisation of molecular interaction networks and biological pathways. However, where the two differ is the fact that Cytoscape is a fully desktop oriented application. It uses Java to visualise the data and thus users needs Java runtime installed to run it. Whereas Cytoscape.js is a JavaScript based library and thus is fully functional in modern web browsers without the need for plug-ins. Although both started as visualisation tools in the medical field, Cytoscape and Cytoscape.js are now libraries which provides a basic set of features for all data analysis, including visualisation. Cytoscape.js supports all major browsers.

The performance of Cytoscape.js will, similar to D3.js, diminish as the number of elements increases[35]. The performance loss comes in terms of browser responsiveness and smoothness of the animations. To illustrate a case: operating on a graph network of roughly thousand nodes barely produces stutter, however working on a network of ten thousand nodes produces stutter in the range of a couple hundreds milliseconds[36]

ArborJs[37] is a jQuery based library. Arbor's simplicity is also its strong suit. Rather being an all-encompassing framework, Arbor.js will only handle the physics math that makes the layouts possible and leaves the rest (data and style) to the developer. The developer can adjust the style with canvas, SVG or HTML.

Thus, we are inclined toward choosing for D3.js, as it is a well-established library for making complex visual representation of data. Furthermore, the compatibility of D3.js and large amount of data seems reasonable. We believe that this overlaps the client's wishes.

[28] Zach Nation. *3 Steps to Scalable Data Visualization in React.js & D3.js*. URL: https://www.codementor.io/znation/3-steps-to-scalable-data-visualization-in-react-js-d3-js-8t7kjxnk5 (visited on 03/05/2019).

[29] *D3.js - Performance Test*. URL: http://tommykrueger.com/projects/d3tests/performance-test.php (visited on 03/05/2019).

[30] https://philogb.github.io/jit/

[31] https://www.slideshare.net/philogb/javascript-infovis-toolkit-create-interactive-data-visualizations-for-the-web

[32] https://zoomcharts.com/

[33] https://zoomcharts.com/developers/en/overview/supported-browsers.html

[34] https://cytoscape.org/

[35] http://js.cytoscape.org/#performance

[36] https://cytoscape.org/js-perf/

[37] http://arborjs.org/

7 | Method of Working

This chapter will explain some of the working methods to give an idea of how the the project is carried out.

7.1 Contact with client

It is very important to define the project according to the company stakeholders' requirements. For this reason we keep continuous and regular contact with the client via several platforms, including WhatsApp, mail, Skype and face-to-face meetings. Additionally, our workplaces are in close proximity to the stakeholders, so emerging questions can be asked and answered quickly.

7.2 Contact with coach

To keep the coach in the loop, a weekly meeting has been scheduled. During these meetings the progress of the project and the extent to which the educational interests of the TU Delft are served are discussed. The coach is kept up-to-date on our deliverables and provides iterative feedback via mail.

7.3 Contact with group

In principle, the group meets from Monday until Friday from 9:00am until 5:30pm to work on the project. This ensures that everyone puts the necessary hours into the project and makes it easy to work together on certain topics and start discussions when necessary.

7.4 Schedule

To give an overview of all the deadlines and milestones, a schedule was created. This schedule will be a dynamic document that will be updated as the project continues.

7.5 Productivity Tools

Git. Not only is Git[1] the de facto productivity tool in the software industry, but we also have acquired ample experience with it at the TU Delft. Git offers several benefits, including version and branching control.

Microsoft Azure. Microsoft Azure[2] is a cloud based computation solution for various applications. A feature of Microsoft Azure is DevOps, which provides continuous integration (CI), continuous delivery (CD) and continuous deployment, which is a combination of the former two.

Trello. Trello[3] provides a neat overview of the tasks to be done, the tasks which are being worked on and the tasks which are already finished. We will work with Trello mostly at the start of the project and migrate to Azure DevOps later on, which will provide similar functionalities.

Document Management System. DMS is used within *Company* to work on documents collectively, securely and iteratively. The documents in DMS are stored on their servers and can only be accessed by employees which are permitted access. DMS stores which employee made an edit and earlier versions of documents are available for viewing. Since permissions to a shared DMS folder had to be configured first, we used Google Drive in the first week, and migrated existing files to DMS later.

[1] https://git-scm.com/
[2] https://azure.microsoft.com/
[3] https://trello.com/

8 | Conclusion

Company asked for an interactive knowledge base of specific data that is scattered over multiple systems. This knowledge base should at least output the following:

1. Who is involved in a topic.
2. Which projects have been started around this topic.
3. What the status on these projects is.
4. Where these projects are carried out (e.g. which home market, which business unit).

Several additional requirements that should be implemented are, among others, a visual graph linking projects to users and a generic solution for adding new data sources as input. Although the visualisation of the data is not a minimal requirement, the addition of such visualisation will significantly increase the value of the product for the client.

In interest of privacy, we should also implement functionality which allows employees to grant permissions to other employees to view certain data. This functionality should be extended with the ability of synchronisation between users and the AD groups. Paired with the requirements come several challenges (chapter 2). The most comprehensive tasks are the processing of sensitive data (section 2.3) and providing a solution for generalised data sources (section 2.7).

With respect to the development technologies, three main choices had to be made; the front end, back end and visualisation frameworks. Because no constraints were imposed by the client, we made choices based on flexibility and ease of use, ultimately landing on the *Vue* front end framework, *Ruby and Rails* for the back end and *D3.js*, a popular data visualisation and manipulation library.

To complete this project, we will work following the milestones stated in 5. Outlining the milestones will provide a structured workflow and inform the client with a preliminary timetable. A couple important milestones to mention would be the Proof of Concept, Minimal Viable Product and the Generic product in weeks 3, 6 and 8 respectively. We plan on working at *Company* office in Rotterdam to ensure close contact with the client. We anticipate with the close contact and our working hours, i.e. Monday until Friday from 9:00am to 5:30pm, to achieve the milestones at the given dates.

Bibliography

[1] S. Alexander. 'Speed Performance Comparison of JavaScript MVC Frameworks (Dissertation).' Bachelor thesis. Blekinge Institute of Technology, 2015. URL: http://www.diva-portal.org/smash/get/diva2:998701/FULLTEXT03.

[2] Bipper Media. *Google Chrome Now More Popular Than IE, Becomes Most Popular Web Browser*. May 2012. URL: https://medium.com/bipper-media/google-chrome-now-more-popular-than-ie-becomes-most-popular-web-browser-8b676c9ae20a (visited on 26/04/2019).

[3] Tolga Bolukbasi et al. 'Quantifying and Reducing Stereotypes in Word Embeddings'. In: (2016). URL: https://arxiv.org/pdf/1606.06121.pdf.

[4] A. A. Bushkin and S. I. Schaen. *The Privacy Act of 1974: a reference manual for compliance*. System Development Corp., 1976.

[5] Chun-houh Chen, Wolfgang Härdle and Antony Unwin. *Handbook of Data Visualization*. Springer, 2016.

[6] D. Clegg and R. Barker. *CASE Method Fast-track: A RAD Approach*. CASE method. Addison-Wesley Publishing Company, 1994. ISBN: 9780201624328.

[7] Tyler Crawford and Tauqeer Hussain. 'A Comparison of Server Side Scripting Technologies'. In: *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2017, pp. 69–76.

[8] *D3.js - Performance Test*. URL: http://tommykrueger.com/projects/d3tests/performance-test.php (visited on 03/05/2019).

[9] Andre Esteva et al. 'Dermatologist-level classification of skin cancer with ...' In: (Feb. 2017). URL: https://www.nature.com/articles/nature21056.

[10] Stephen Few. 'Data visualization pas, present and future'. In: (2007). URL: https://www.perceptualedge.com/articles/Whitepapers/Data_Visualization.pdf.

[11] Mike Gouline. *Library-driven development*. Sept. 2015. URL: https://blog.gouline.net/library-driven-development-992b561eb5f1 (visited on 26/04/2019).

[12] *JavaScript Environment Requirements – React*. URL: https://reactjs.org/docs/javascript-environment-requirements.html (visited on 02/05/2019).

[13] Glenn E. Krasner and Stephen T. Pope. 'A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80'. In: *J. Object Oriented Program.* 1.3 (Aug. 1988), pp. 26–49. ISSN: 0896-8438. URL: http://dl.acm.org/citation.cfm?id=50757.50759.

[14] Michael J. Mcguffin. 'Simple algorithms for network visualization: A tutorial'. In: *Tsinghua Science and Technology* 17.4 (Aug. 2012), pp. 383–398. DOI: 10.1109/tst.2012.6297585. URL: https://ieeexplore.ieee.org/document/6297585.

[15] Steven R. Meier. *Building and Managing an Effective Project Team*. Sept. 2008. URL: http://connection.ebscohost.com/c/articles/34409947/building-managing-effective-project-team.

[16] Judith Moeller and Natali Helberger. 'Beyond the filter bubble: concepts, myths, evidence and issues for future debates'. In: (June 2018). URL: https://www.ivir.nl/publicaties/download/Beyond_the_filter_bubble__concepts_myths_evidence_and_issues_for_future_debates.pdf.

[17] Zach Nation. *3 Steps to Scalable Data Visualization in React.js & D3.js*. URL: https://www.codementor.io/znation/3-steps-to-scalable-data-visualization-in-react-js-d3-js-8t7kjxnk5 (visited on 03/05/2019).

[18] Eric Wohlgethan. 'Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js'. Bachelor thesis. Hochschule für Angewandte Wissenschaften Hamburg, 2018.