VRIJE
UNIVERSITEIT
BRUSSEL

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

EDITABLE AND CUSTOMIZABLE KNOWLEDGE MAPS

Thijs Spinoy

June 2019

Promotor:
Prof. Dr. Olga De Troyer

Advisor:
Jan Maushagen

Sciences and Bio-Engineering Sciences

Abstract

Samenvatting

Acknowledgements

Declaration of Originality

Contents

List of Figures

Listings

List of Tables

1 Introduction

The more people can store in their memory, the less they have to look up in external sources and the faster they can work. However, processing and storing a big amount of information in the human memory for later reuse is not easy, especially when the information is presented in the form of a long text. Therefor, students for example, make schemes and summaries of their study material. The reason why they do this is to make it is easier to learn and remember the material. Schemes and visualizations make relations between pieces of information explicit and are easier to grasp. In addition, most people are visual learners, meaning that they learn better when using visual aids. Not only the way of learning new subject material, but also other activities can be supported by means of a visualizations. For instance, if you write down the structure of a computer program in words, it is more difficult to discuss that structure with other people than when the structure is expressed by means of a diagram. This can be explained as follows.

According to Moody (2009), there exists a difference between visual notations and textual languages in how they encode information and how they are processed by the human mind: textual representations are one-dimensional and are processed serially by the auditory system, while visual representations are two-dimensional and processed in parallel by the visual system.

Further, the form of representations/visualizations has a greater effect on understanding and problem solving than their content (Moody, 2012). In other words, for a visualization, the way the content is represented is more important than the content itself.

Moody (2009) also defines the term *Cognitive effectiveness* as “the speed, ease and accuracy with which a representation can be processed by the human mind”.

The better the cognitive effectiveness of a visual notation, the better human communication and problem solving can be done.

The goal of this thesis is to create a tool with high cognitive effectiveness, in which it is possible to represent linked data and knowledge in a visual manner. It should not only be possible to create a visual notation of the data, but also to edit the data as well as extending the representation with additional data. The solution is based on a visualization tool created by Janssens (2013), called GuideaMaps. This application was mainly built to provide support for the requirement elicitation for serious games. It provides the functionality to enter information in pre-defined maps (trees of nodes). We present a new version of GuideaMaps, which can be used for other purposes as well and which has a bunch of interesting improvements under the hood.

1.1 Problem Statement

For the first version of GuideaMaps, the main goal was to develop the following:

> “A tool that allows the different people (and with different background) involved in the development of a serious game (e.g., against cyber bullying) to brood over the goals, characteristics and main principles of a new to develop serious game. The tool should be easy to use and usable in meetings. Therefore, we want to explore the characteristics and capabilities of a tablet (i.e. iPad).” (Janssens, 2013)

By specifying the goal in this way, end users of the application are restricted in different ways. First, they need an iPad to be able to use the application. Another type of tablet with a different operating system is not possible, because GuideaMaps was created and designed for iOS only. If someone doesn’t have access to an iPad, (s)he cannot use the application, which is a hard restriction.

Furthermore, the tool focuses on requirement elicitation. Initially, the tool was created for the purpose of requirement elicitation for serious games, but it can also be used for the requirement elicitation in other domains (De Troyer & Janssens, 2014). However, because the visualizations are based on pre-defined templates, the nodes can only be edited by the end-user in a limited way: content can be given and the background color can be changed, but the end-user cannot add new nodes. In addition, the visual notation used is fixed: the creators of the visualization templates cannot edit the representation of a node or define their own representation (e.g. change the length and width or use a different shape). Not being able to do this is a limitation in the sense that for some purposes this default visualization may not be very suitable. Furthermore, for defining a template the author had to use XML and no graphical editor was available for this purpose making it harder for non-ICT schooled people to define new templates.

1.2 Research Goals

The issues discussed in the previous section indicate that the first version of GuideaMaps comes along with some limitations. Therefore, the following research goals for a new version of GuideaMaps were formulated:

Goal 1
The new version of GuideaMaps should work on all common devices and on different operating systems.

Goal 2
It should be possible to pre-define the maps, i.e. the templates, in a graphical way.

Goal 3
The new version of GuideaMaps should allow the end-user to extend and modify the pre-defined map in some restricted way.

Goal 4
The application should be generic in such a way that it can be customized to be usable for different purposes, i.e. the user should be able to define its own graphical representation for the visualization.

This thesis presents a solution, called *GuideaMaps 2.0* (?), for the problem statement taking the research goals into account. How the tool achieved the formulated research goals is explained into detail in the rest of this thesis.
TODO: Voeg stuk toe dat de structuur van de thesis uitlegt.

2 Related Work

TODO: ook kijken naar bestaande tools.

There exist lots of ways to visualize data and the relations between data. In this chapter, we discuss work related to the needs of GuideaMaps 2.0.

2.1 Mind Maps

The most well-known technique to visualize related data is to create a mind map (a.k.a. idea map). This technique is mainly used to show the relation between portions of information and for brainstorming purposes. Other applications where this technique is used are note-taking, and problem solving. (Balaid et al., 2016)

Mind maps are created by writing the main idea in the middle of the drawing, while all sub-ideas are placed around that center node. Each sub-idea is connected with its parent by means of a line. Hence, this kind of visualization is not difficult to create or understand. Its simplicity is one of the reasons why it is used a lot in practice.

Because mind maps is not a new concept, but one that most people already know quite well, we will only discuss one important aspect about this visualization technique. When using a digital version of mind maps, in general, the user can change the position of the nodes. Wiegmann et al. (1992) state that maps taking the gestalt principles into account would be more performance-effective than maps that don't integrate gestalt principles. Therefore, digital mind map systems usually allow their users to re-organize the nodes using drag and drop and by changing the color of the nodes. In this way, the user can, for example, put nodes containing similar data closer to each other and give them the same color (proximity and similarity principle).

2.2 Visual Metaphors

Another technique to represent content is visual metaphors.

> "A visual metaphor is a graphic structure that uses the shape and elements of a familiar natural or manmade artefact or of an easily recognizable activity or story to organize content meaningfully and use the associations with the metaphor to convey additional meaning about the content." (Eppler, 2006)

This technique could be used if you want to help users to memorize the most important elements of a topic, method or concept. In that case, you have to choose a metaphor which has properties in common with the topic, concept or method. (Eppler, 2006) An example[1] of a visual metaphor is shown in Figure 2.1.

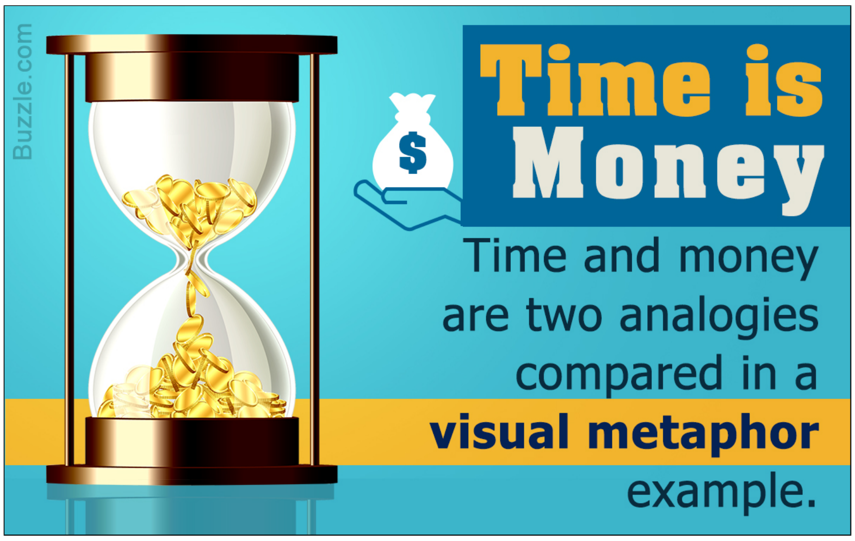

Buzzle.com
Time is Money
Time and money are two analogies compared in a visual metaphor example.

Figure 2.1: Example of a visual metaphor.

As you can see in the figure, the visualization is completely adapted to the specific case of the topic. In this way, it is very difficult to create a generic metaphor, which can be used for different topics. As our tool should be usable for different topics, the technique of visual metaphors is not suitable for the needs of our tool.

2.3 Knowledge Maps

According to Balaid et al. (2016), *knowledge maps* is an umbrella term for tools and techniques like mind maps. O'Donnell et al. (2002) defined the concept as follows:

> "Knowledge maps are node-link representations in which ideas are located in nodes and connected to other related ideas through a series of labeled links."

[1]`https://media.buzzle.com/media/images-en/illustrations/conceptual/1200-609974-visual-metaphor.jpg`

This way of representing information has for example a positive impact on students. The paper of O'Donnell et al. (2002) teaches us that students using knowledge maps are better in remembering the main ideas of the subject in comparison to the ones that study from the text without the visualization. As GuideaMaps's purpose is more focused on representing large amounts of knowledge in an easy to grasp way, remembering the content is less important. However, the node-link representation with the main idea in the center also showed to be useful for this purpose, as illustrated by the popularity of mind maps.

Next to mind maps, concept maps is a second technique included under the umbrella of knowledge maps. Concept maps are in some sense similar to mind maps but they do have some different characteristics. First, the purpose of a mind map is to associate ideas, topics or things, while concept maps illustrate relations between concepts. Further, the structure of a concept map is mostly hierarchical and visualized like a tree. On the other hand, mind maps sometimes have a radial layout and not hierarchical. (Davies, 2011)

Hence, we can state that GuideaMaps makes use of a knowledge map visualization and more specifically some kind of combination of mind maps and concept maps.

3 User Classes & Requirements

In general, an application should meet a lot of requirements in order to deliver some quality to the users. A system should provide the right kind of functionality, but it should also be *usable*, i.e. easy to learn and easy to use. Below are two definitions of usability which are often used:

Definition 1
Usability is a measure of the ease with which a system can be learned and used, its safety, effectiveness and efficiency, and attitude of its users towards it. (Preece et al., 1994)

Definition 2
Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. (ISO, 1998)

Note that the second definition emphasizes the fact that usability is dependent on the target users and on the context of use. This means that one system can be usable for one type of user but not for another type of user or usability in one context but not in another context. Therefore, we also have to consider the different users of a system. In general, the users are classified into user classes. Users in a user class are similar in terms of their characteristics and how they use the system.

Hence, we can distinguish between functional requirements and usability requirements. The system should meet the functional requirements to provide the right functionality and it should meet the usability requirements to be usable. There are also requirements that do not belong to either category, e.g. the so-called non-functional requirements.

This chapter starts by identifying the different user classes. Next, we will present and justify the major requirements formulated for our system, functional as well as usability requirements and other requirements. Detailed requirements and requirements that are rather straightforward are omitted here.

3.1 User Classes

TODO: Beschrijf de verschillende user classes.
A map creator has the rights to change the structure of the visualization, while the end user is restricted in which data he can edit.

3.1.1 Functional Requirements

1. **Customization**
   Goal 3 formulated earlier stated that the new version of GuideaMaps should allow the end-user to extend and modify the pre-defined map in some restricted way, i.e. they should be able to change the background color of nodes, to add child nodes and to remove nodes they don't need, and it should also be possible to add new options in choice nodes. However, end-users should only be able to delete nodes they added themselves and not the nodes defined by the map creator. Otherwise it would be possible for end users to change the pre-defined map completely. The purpose of allowing the end-user to edit the map in some restricted way is to adjust it to situations that were not foreseen by the map creator.

2. **Modes And Rights**
   Because we have two user classes that require different functionality, two modes are needed: an end user mode and a map creator mode. These modes can be seen as a security mechanism to not let a particular end-user mess with a map definition.

3. **Zoom the visualization**
   Zooming in or out such that you get less or more information at the same time on the screen is a frequently provided functionality for large visualizations. A feature to zoom is, for example, very useful in situations where you want to compare different parts of the visualization or focus on a certain part. The scrolling gesture is probably the best gesture for this action, because this is a well-known way to zoom in applications (e.g. Google Maps). Using the same gesture as in other applications will improve the learnability of our tool.

4. **Zoom to fit**
   A variation on the zooming feature is "zoom to fit" (a.k.a. zoom until the complete figure fits into the bounding box). With custom zooming, the user can set the zooming level to meet its needs. Zoom to fit automatically adapts the zooming level and moves the content of the application until everything fits on the screen.

5. **Genericity**
   Goal 4 formulated earlier states that the application should be usable for different purposes. While GuideaMaps was usable in the context of domain specific requirement elicitation, our tool should also be useful in many other cases. Therefore, some requirements concerning the genericity of the application are needed:

   (a) The tool should be generic in such a way that it is possible to use a different representation, e.g. shape/size for the nodes and the links.

   (b) A developer should be able to create its own implementation for the nodes and the links, which then can be *plugged in* into the system. Leg het verschil uit met a.

6. **Simultaneously Edit Same Visualization**
   Multiple users should be able to work on the same visualization at the same time.

3.1.2 Usability Requirements

In order to obtain a high usability, we should formulate the necessarily usability requirements for the application. Later in this thesis, when we explain the implementation details, we will come back to these requirements and discuss how we managed to meet them.

1. **Intuitiveness**
   It is important to keep in mind that the tool should not only be used by people with experience in Computer Science. It does not matter whether or not the user has a background in Computer Science, he should be able to easily learn to use the system in a short time. Therefore, design choices have to be made carefully.

   (a) As icons[1] take in general less space than text, we will use icons to indicate possible actions. Some examples of these actions are (1) adding a child node, (2) viewing the content of a node, (3) editing a node and (4) expand/collapse a node. However, it is important to choose for clear, not misunderstandable icons. The icons should not be ambiguous, they should link to one particular action and thus the user should know exactly what to expect when clicking on the icon. Well chosen icons are one of the factors in the design that contribute to learnability and ease of use.

   If the user knows it is not possible to add child nodes, we would state that a plus- and minus-icon are also possible to indicate the possibility to expand or collapse a node. If the visualization allows to add child nodes, the plus-icon should be used for this action and not for the expand- or collapse-action. TODO: Niet duidelijk, beter uitleggen.

[1] `https://fontawesome.com/`

(a) Plus (b) Explore/Edit (c) Expand (d) Collapse

Figure 3.1: Good icons for the following actions: (a) add child node, (b) explore and edit node, (c) expand node and (d) collapse node.

TODO: De icons in de figuur zijn al de oplossing, niet de requirement. Plaats ze ergens anders.

(b) As already mentioned in Functional Requirement 3, gestures for common actions should not differ from the gestures used for the same action in other applications.

2. **Learnability**
   TODO: Dit is geen echt usability requirement. Moet beter.
   Intuitiveness and learnability are somehow related to each other. It is easier to learn how a system works if the possibilities are intuitive and actions are not hidden.

3. **Efficiency**
   TODO: Dit is geen echt usability requirement. Moet beter.
   As a user, it should be easy to achieve your goals. The application should be created in such a way that user errors are (almost) impossible.

4. **Accessibility**
   The target audience of the application should be limited as little as possible.

   (a) We don't expect the user to have a background in Computer Science or visualization techniques. TODO: Dit is geen echt usability requirement. Moet beter.

   (b) Color blind people should still be able to use the system. We can make sure this is possible by allowing users to choose for colors that are distinguishable by color blind people. TODO: Dit is al een deel van de oplossing. Plaats de oplossing ergens anders.

3.2 Other Requirements

1. **Device- and OS-independent**
   According to Research Goal 1, the application should run on different kinds of devices (e.g. tablets, laptops and desktops) and operating systems (Android, iOS, MacOS and Windows). While version 1.0 was only designed for an iPad, our version should provide a solution for this restriction.
   The only restriction on the used device is that it needs to have a screen that is large enough because it is not very convenient to work with the visualization on small screen areas, e.g. on smartphones. The application could run on smartphones but it is not recommended nor required to use it on devices with relatively small screens.

2. **Code Separation**
   The core of the application should be completely separated from the parts that are customizable for the end-users.

4 Implementation

TODO: Op voorhand al beter beschrijven wat de applicatie moet doen. Dit is mogelijk nog niet duidelijk.
The previous sections explained *what* the application should do, which goals should be achieved and which requirements we surely want the application meets. In this section, more details are provided about *how* all this is translated to an implementation.

4.1 Design Choice

Before we started with the actual implementation, we had to decide how to achieve Research Goal 1, i.e. how will we make sure the application is device- and OS-independent? A first technique could have been to make use of the Java Virtual Machine. This approach would work but we want to make it possible to let multiple people work on the same visualization (Functional Requirement 6, eventually at the same time. Therefore, it is more convenient to have a browser-based application.TODO: cross-platform tools was ook een optie geweest. Waarom niet?

Nowadays, a lot of browsers exist, each with their own characteristics and differences. Because the goal of this thesis was to have a functional prototype, it is not required that the application works perfectly on *all* possible browsers. However, we made sure that the tool works without any issues in Google Chrome. We chose for Google Chrome because this is one of the most widely used browsers in the world.

4.2 Technologies

A lot of technologies exist to create web applications and visualizations. A number of technologies assisted in the developing process of such an application. In the following subsections we present the selected technologies, as well as why we chose for this particular technology and not for the alternatives.

4.2.1 ReactJS

ReactJS[1] is an open source library to create user interfaces. One of its main goals is to provide the best possible rendering performances[2]. Performance is good because ReactJS allows developers to break down the user interface into different components. Each component has its own *state*, which contains information about the content of the component. This state can be updated while the application is running and if such an update is made, only this component is re-rendered instead of re-rendering the complete UI[3]. Hence, this provides a huge benefit for the performance. Next to that, it is also not very hard to learn to code in React comparing to other frameworks (e.g. AngularJS). If the developer knows HTML and JavaScript, he will be able to code in ReactJS quickly.

Because of these benefits, ReactJS is the framework in which the GuideaMaps application is implemented. Each node and each link is considered as a separate and unique React Component. The most important reason for implementing the nodes like this is performance: if the state of the node is updated, only this node is re-rendered and not the complete UI.

Alternatives for ReactJS are for example AngularJS and VueJS[4]. ReactJS is by far the most used of these three. Further, it is said to be easier to learn because you only have one structure ("Component"), while in AngularJS this is not the case. Also, VueJS lacks resources and is not used a lot in practice, which makes it more difficult to discuss problems with other developers. Maybe the most important benefit of ReactJS is its performance; React is really fast and our tool should have fast rendering times as well. Hence, because the alternatives seem to have some important drawbacks, we chose for ReactJS as the framework for creating the application.

4.2.2 d3

Another helpful tool is d3. With d3-hierarchy[5], it is possible to transform JSON-data into hierarchical data. Having this kind of data makes it much easier to create a tree- or cluster-structure. In the case of GuideaMaps, a clustered

[1]`https://reactjs.org/index.html`
[2]`https://medium.com/@thinkwik/why-reactjs-is-gaining-so-much-popularity-these-days-c3aa686ec0b3`
[3]`https://facebook.github.io/react/docs/why-react.html`
[4]https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d
[5]`https://github.com/d3/d3-hierarchy`

visualization is very useful. The *main*-node (a.k.a. the root node) is then positioned at the center, such that its child nodes can be placed around it. Hence, the further a node is away from the center, the lower it is in the hierarchy. Also, the visualization will not be messed up by positioning the nodes in this way, because every node has exactly one parent. This means there will not be a spaghetti of links where you cannot see from which node the link comes and to which node it is pointing.

Further, d3 makes it easier to create a zoomable layout. The tool is able to update the positions of the nodes each time the zooming level is changed. Hence, as a developer, you do not have to take care of the updated positions if you make use of d3. By using this technique, the functional requirements about zooming (section 3.1.1) are easily achieved.

The alternatives for d3 most of the time come with a problem d3 does not have. For example ChartJS and ChartistJS are limited in the number of features: you can create nice charts with it, but d3 provides more visualizations than only charts. Further, some of the alternatives are commercial (Highcharts, Webix). d3 is open-source and provides functionality to zoom, to create a cluster of nodes based on JSON-data describing these nodes, etc. Because of the wide range of possibilities with d3, the fact that it is widely used and supported by all modern browsers and that it is able to act together with ReactJS, we believe that d3 was a better choice than its alternatives.

4.2.3 Tailwind CSS

Nowadays, the "look and feel" of an application is very important. Everything should look pretty and, as already mentioned in section 3.1.2, the actions should be straightforward and visible. Implementing a nice style can require a lot of code. The code for these styles can become a big part of the implementation. Hence, a good framework is necessary to reduce the lines of style code to a reasonable number and to help to improve the readability of the code.

Tailwind CSS[6] is a framework that assists developers to style their application. The difference with more famous frameworks, like Bootstrap, is that Tailwind CSS has no default theme. If you want to use a Bootstrap-feature, this eventually comes along with other features you do not always want and it can be quite hard to undo the part you do not want. Furthermore, you have to write additional lines of style code to undo the unwanted parts. With tailwind on the other hand, you can grab only the features you want, without side-effects. Figure 4.1 shows an example with two small listings. The first uses inline style while the second makes use of tailwind CSS.

[6] `https://tailwindcss.com/`

```
<div
  style={{
    position: absolute,
    border: 1px solid black,
    borderRadius: 0.25rem,
    padding:: 0.5rem,
  }}
/>
```

Listing 4.1: Normal CSS, no tailwind.

```
<div
    className={
      'absolute border
       ↪  border-solid
       ↪  border-black rounded
       ↪  p-2'
    }
/>
```

Listing 4.2: With tailwind CSS.

Figure 4.1: Difference when using tailwind CSS or not.

The figure illustrates the difference to implement four CSS property-value pairs in normal CSS and implementing the same four with tailwind. In the case of normal CSS, we need four lines of code to obtain the intended result. On the other hand, with tailwind CSS, we add some classes providing the same style. The classnames can be placed on a single line instead of four. This example proves that the number of lines can be decreased. TODO: beter uitleggen, want het lijkt alsof het even veel lijnen zijn nu.

Bootstrap can be very interesting to use in applications and websites that should run on devices with small screens (e.g. smartphones). But we already decided not to optimize our application for such devices (3.2). Given the example and the fact that Tailwind CSS does not have a default theme, we prefer Tailwind over Bootstrap. Keep in mind that it is certainly possible to create the same application with Bootstrap, but with Tailwind CSS, the code will be easier to understand.

4.3 Main Code Structure

With the technologies mentioned in the previous section, the most important pillars the application relies on are discussed. In this section, we will explain the main structure of the code, such that it is clear how all elements work together. Figure 4.2 shows a visualization of the structure of the code.

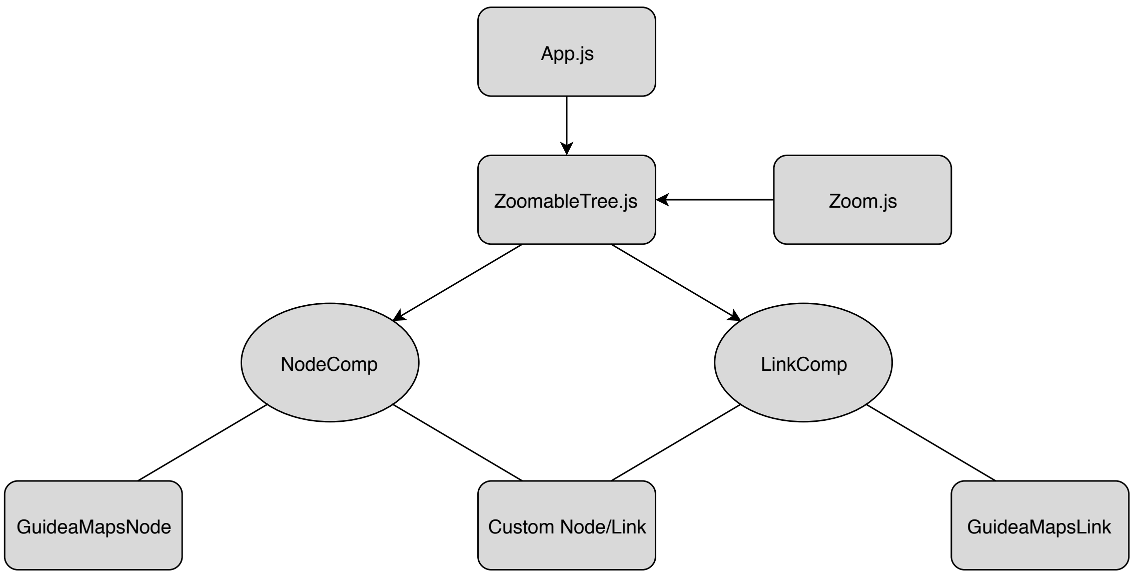


Figure 4.2: Structure of the application.

The application is developed in such a way that it can be used as a library for other purposes than GuideaMaps. In this way, Research Goal 4 to have a generic solution will be achieved. The general, always-returning part of the code can be found in App.js, ZoomableTree.js and Zoom.js, where the layout of the nodes is defined as well as the implementation to allow the user to zoom the visualization in and out. To realize the library, the layer of the NodeComp and LinkComp is very important. For GuideaMaps (shorthand GM), an implementation for NodeComp and LinkComp, being GMNode and GMLink is provided. Each implementation describes how every node and every link should look like in the visualization. When a particular user would like to have a different representation for the nodes or the links or both, new components (e.g. MyCustomNode and MyCustomLink) should be implemented. In the rest of the code, only one line should be adapted: in App.js, ZoomableTree is called with a certain number of props. Two of these props are NodeComp and LinkComp, which are set to GMNode and GMLink, respectively, by default. Hence, the only action that is required to *plug in* an other component is replacing GMNode and GMLink by this component (e.g. MyCustomNode and/or MyCustomLink). Hence, the visualization can be customized to the needs of the user and Functional Requirement 1 is achieved. Figure 4.3 shows the part of the code in App.js that should be adapted as explained. Note that the shown props are not the only props that are passed to ZoomableTree. The others are omitted for readability.

```
<ZoomableTree
    NodeComp={GMNode}
    LinkComp={GMLink}
/>
```

```
<ZoomableTree
    NodeComp={MyCustomNode}
    LinkComp={MyCustomLink}
/>
```

Listing 4.3: Default components. Listing 4.4: Custom components.

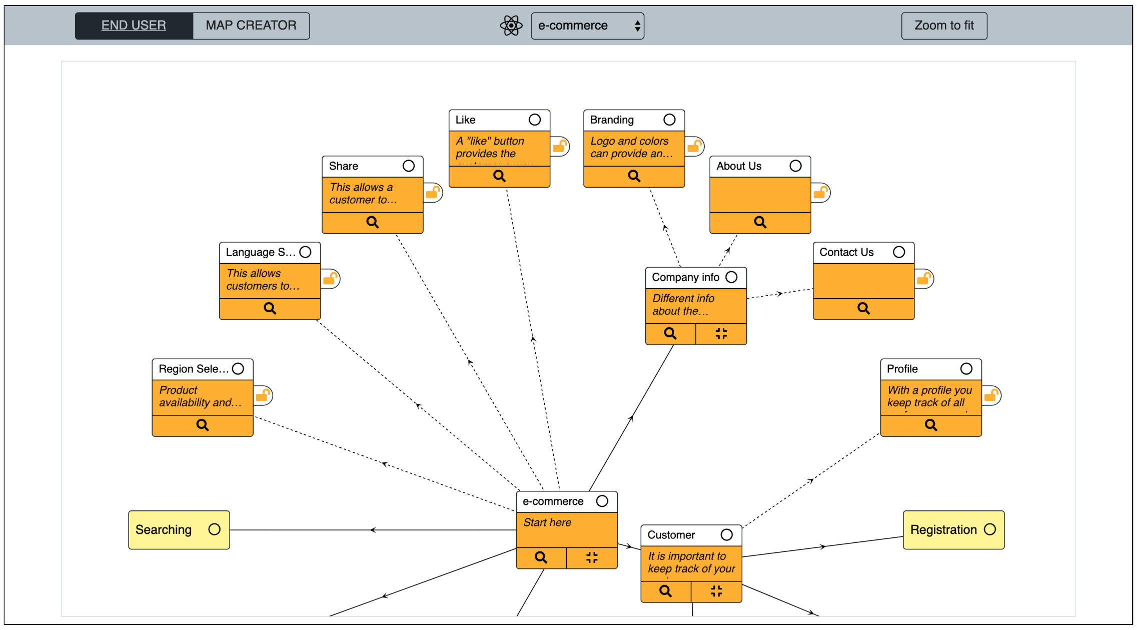
Figure 4.3: Two listings showing how to use the library.

The reason why we chose for this approach is that now the developer does not have to change anything of the default implementation. He only has to create his own components and plug them in as props for ZoomableTree and leave the rest of the implementation as it is.

Next to custom components, a developer can also implement his own functions to handle changes in the visualization. For example, to add a new child node, GuideaMaps calls the function passed to the *onAddNode*-prop (i.e. addGMChildNode). In a custom implementation, you can pass another function to this prop to make sure that other work is done. If you do not want to allow users to add child nodes, you do not have to remove this line but you just replace addGMChildNode by *() -> null*, a function that does not do anything (because the library is created in a way to avoid changes in the default code structure). More details about the use of the library can be found in Chapter 5, where a complete use case is elaborated.

4.4 Default Implementation: GuideaMaps

Now the overview of the structure of the application has been discussed, we will consider the GuideaMaps visualization and its implementation details in this section. An example visualization can be seen in figure 4.4.



END USER
MAP CREATOR
e-commerce
Zoom to fit
Like
A "like" button provides the
Branding
Logo and colors can provide an…
Share
This allows a customer to…
About Us
Language S…
This allows customers to…
Contact Us
Company info
Different info about the…
Region Sele…
Product availability and…
Profile
With a profile you keep track of all
e-commerce
Start here
Searching
Customer
It is important to keep track of your
Registration

Figure 4.4: GuideaMaps Layout.

The nodes represent a specific part of the data and the links illustrate the relation between the data (i.e. the nodes). Before we discuss the nodes into detail, we start with the *navigation bar* above the visualization itself. This navigation bar is divided in three equal parts. In the center, the user can select via an option menu which map or template he wants to use. The name of the selected option is always visible. This functionality is available because the user should be able to switch between different tasks. In each task, the user can work with a different map or start a new map based on one of the existing templates, created by a map creator.

The right part of the navigation bar contains a single button. A click on this button makes sure the visualization is zoomed in such a way that all nodes fit in the window. This is one of the reasons why we mentioned in the requirements (section 3.2) that the screen of the used device should not be too small. If the screen is too small, the nodes of a larger visualization will overlap each other because otherwise they would not fit all on the screen. When nodes overlap, a lot of information can be hidden and the visualization can become useless.

The left part of the navigation bar is created to be able to switch between the user modes (i.e. end user or map creator). As required, a map creator has more rights and hence he can perform more actions than an end user. In the following subsections, the differences between these modes are elaborated.

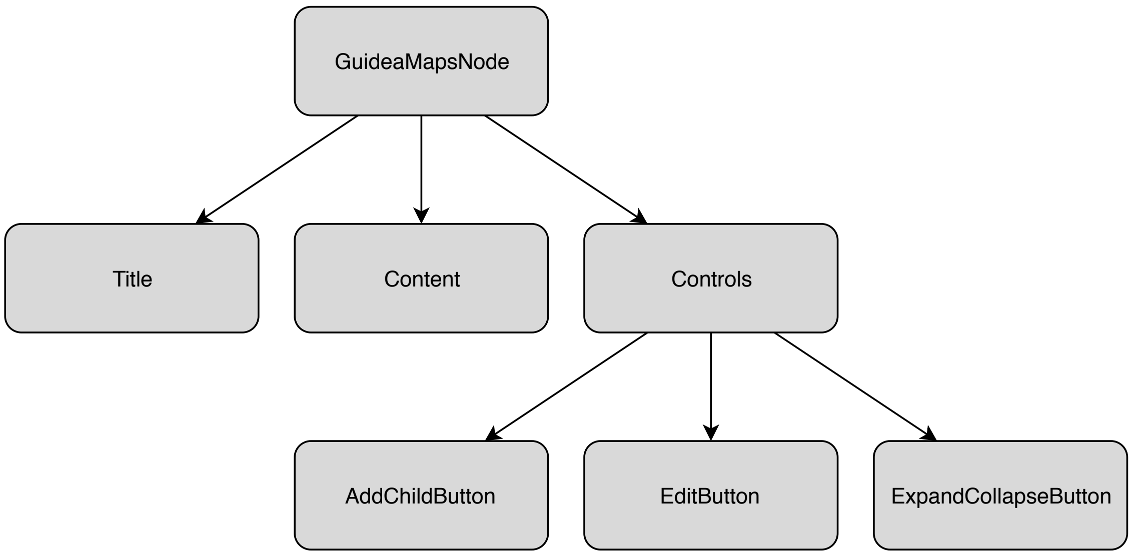


Figure 4.5: Structure of GuideaMapsNode.

As you can see in figure 4.5, the structure of a GuideaMapsNode node is quite simple. Each node consists of three html *div*-elements: a title-div, a content-div and a controls-div explained below.

4.4.1 Title

The title-div is positioned at the top of the node. It consists of the title text when this is given by the map creator. Otherwise, it shows *Insert title* in italics to remind the user that he still has to set a title for this node. An end user can set the title if no title exists yet, but he can never change it. The reason for this is because we distinguish two situations. In the first situation, the end user adds a child node for a particular parent. This node has no title by default and then he should be able to insert a title. In the second situation, the user opens an existing node created by the map creator. In that case an end user is not allowed to change the title of the node, because otherwise he would be able to change the meaning of the node.

Next to the title-text, the title-div contains an icon placed near the right border. This icon indicates to an end-user whether all information for the node, i.e. content, is provided or not. The icon that is shown also depends on whether the information in the child-nodes is given or not. We distinguish three possible situations:

1. The node itself and all of its (non-optional) children are correctly filled in. In this case the icon will be a completely filled circle. TODO: dit is de eerste keer dat het over optional nodes gaat. Volgorde veranderen?

2. The node itself and all of its (non-optional) children are still empty. In this case the icon will be an empty circle.

3. In all other cases, the node itself or at least one of its child-nodes is filled. Then the icon will be a semi-filled circle.

This icon can assist the user in determining whether all information requested by a template is given. For example, if he checks the root node and sees that the circle is completely filled, he knows that all mandatory information in every child-node is given. On the other hand, suppose only one node does not yet contain the required content, then the user starts from the root node and always follows the child-node with a semi-filled circle. In this way, he will find the incomplete node much faster than in the case he has to check all the nodes, one by one. Hence, the icon is an element that helps to improve the usability of the application.

4.4.2 Content

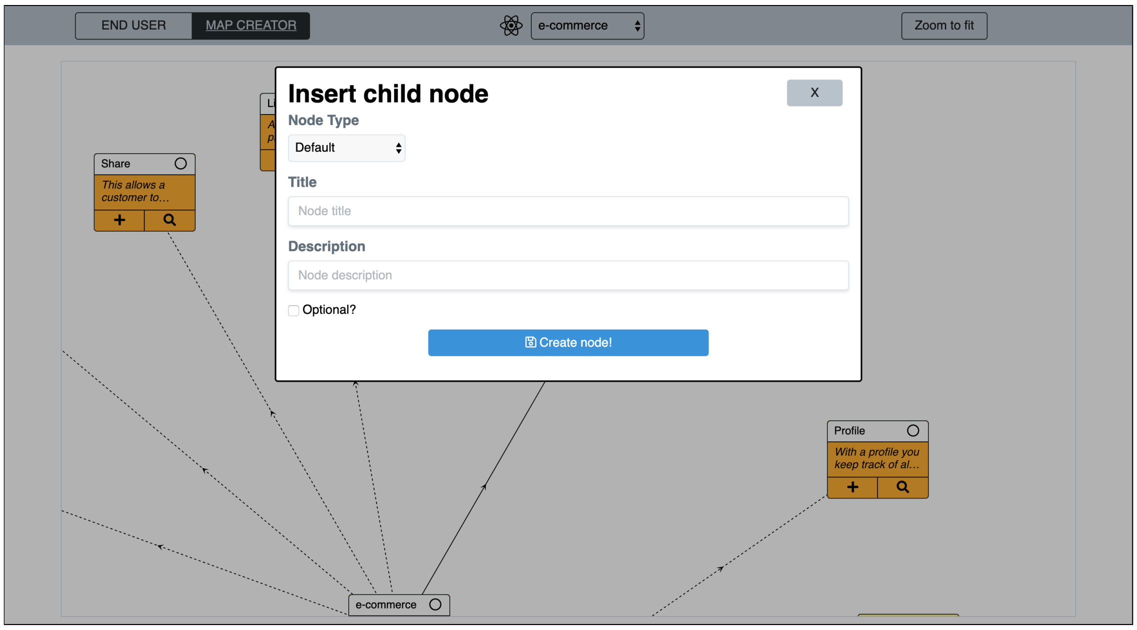
The content of each node is some text describing the information required for the node. As long as no content is provided by the user, a description is shown in italics to instruct the user which content to provide.

4.4.3 Controls

The last part of a node is the controls-div. With *controls* we mean the different actions a user can take concerning the particular node. The number of actions a user can perform depends on the mode. An end user has two buttons: one to “open” the node to view and edit the data, and one to expand or collapse the node, i.e. to show or hide the child nodes, respectively. When a node is collapsed, all child nodes on all lower levels in the hierarchy are hidden. On the other hand, when a node is expanded, only the child-nodes of the next level in the hierarchy are shown.

Because a map creator is allowed to add child nodes, a third button can be found in his controls-div of the node to add a child node. A click on this button opens a small modal, where some information about the new node should be provided before the node can be created. Figure 4.6 illustrates what the modal looks like when the user wants to add a default node. He has to start by selecting the option “Default” at the top, after which two input fields appear to insert a title and a description for the node. A click on the button at the bottom will add a child node of the selected type and with the provided title and description.

It is possible to add an *empty* node if the user does not provide a title and/or a description. Even though it is not really useful to add nodes without any context, we only require the user to select the type of the child node. If the type is unknown, the node will not be added. A situation where a map creator would add an empty node is when he wants to remind himself that a child node is necessary at that place, but details about the expected content are not yet known.



Insert child node

Node Type
Default

Title
Node title

Description
Node description

Optional?

Figure 4.6: The edit modal in map creator mode.

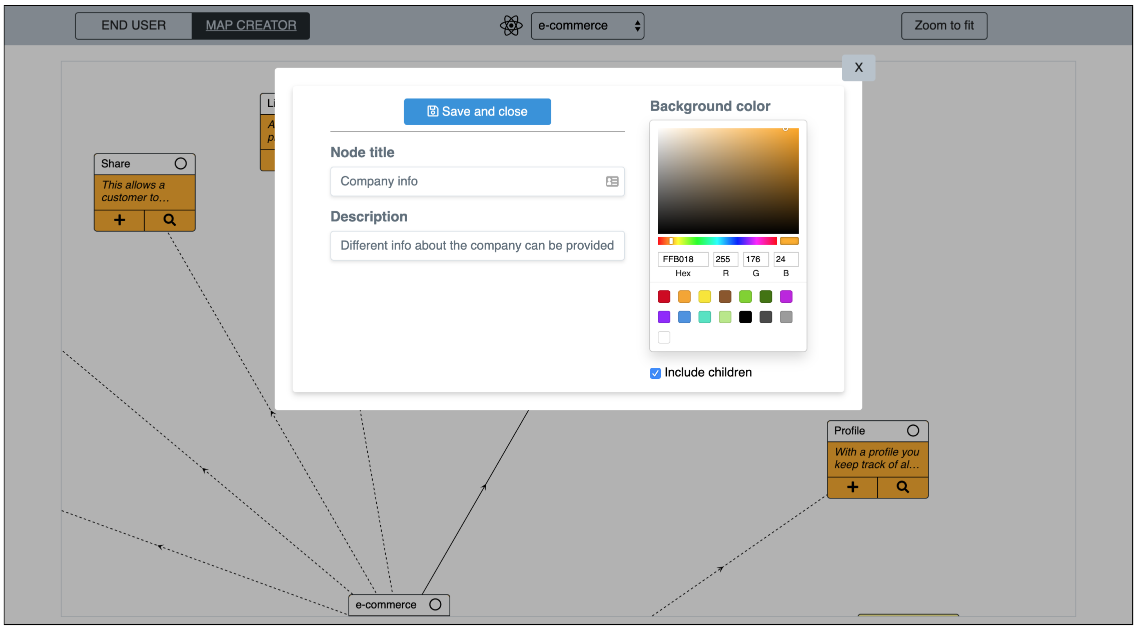
TODO: describe addition of choice nodes

4.4.4 EditModal

A click on the button to open the node opens a modal representing the data. What this modal looks like depends on the mode. If we are in map creator mode, a form will be shown to allow to change the title, the description, and the background color (Figure 4.7). In the case of the end user mode (Figure 4.8), only the content and the background color can be changed. Only the first time a child node is added and no title and description is available yet, the end user is able to fill in these data as well.

When changing the background color, the user can choose to use the color also for all children or not. If the checkbox “include children” is checked, the background color of all child-nodes on all sublevels in the hierarchy will be changed to the new color. Otherwise, only the background color of the current node will be changed.

The text color is black by default. This can become a problem when the background color is changed to a dark color and certainly when it is black as well because then the text is not readable anymore. To solve this issue, we wrote some CSS rules that change the text-color depending on the background color. A black (or dark) background will result in white text and a white (or light) background results in black text.



Node title

Company info

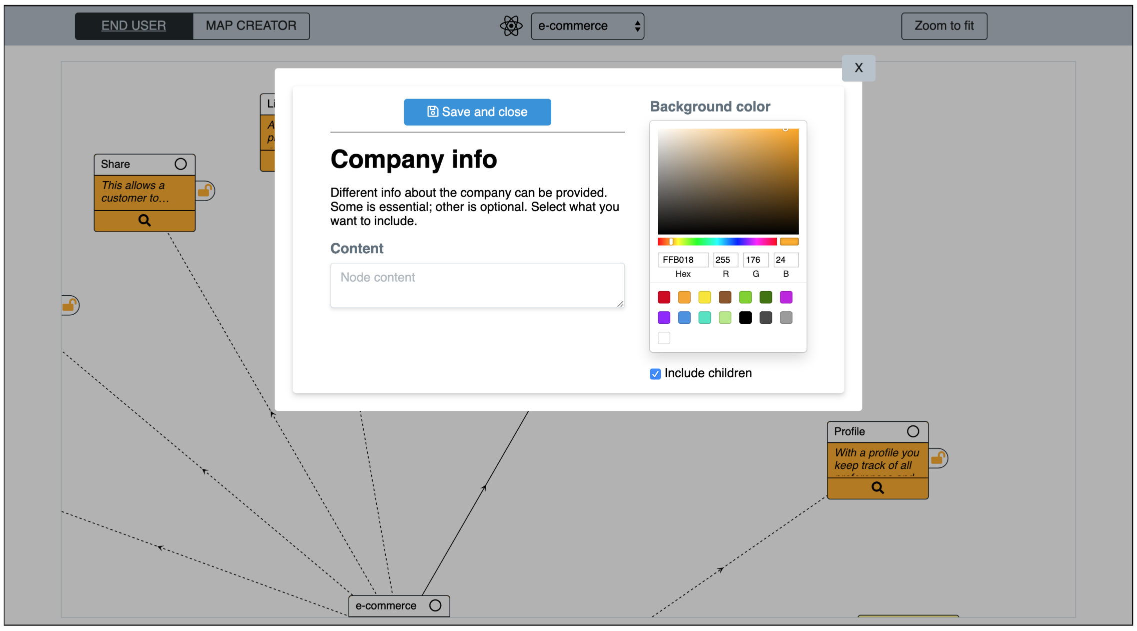
Description

Different info about the company can be provided

FFB018 255 176 24

Include children

Figure 4.7: The edit modal in map creator mode.



Company info

Different info about the company can be provided. Some is essential; other is optional. Select what you want to include.

Content

Node content

FFB018 255 176 24

Include children

Figure 4.8: The edit modal in end user mode.

4.4.5 Optional nodes

The visualization also provides optional nodes. They can be recognized by the representation of the link between this node and its parent. A regular node is connected to its parent via a solid line, while an optional node is connected via a dashed line. A special characteristic of optional nodes is that these nodes can be disabled by the end user. Therefore, optional nodes contain an additional button, represented as a lock, at their right side. As long as the node is enabled, the button is represented by an open lock. A click on this button disables the node and changes the icon to a closed lock. The node can be enabled again by clicking the lock again. Next to the lock icon, a disabled node can be recognized by its gray background and a blur so that the content is less readable.

5
Another Use case

To demonstrate that we achieved Research Goal 4, we implemented a different use case, called *Plateforme DD*. The goal was to visualize the links between the content of the website[1] TODO: meer uitleg geven over de bedoeling.. The visualization and functionality is completely different from the one for GuideaMaps. The visualization for the map can be seen in Figure 5.1.

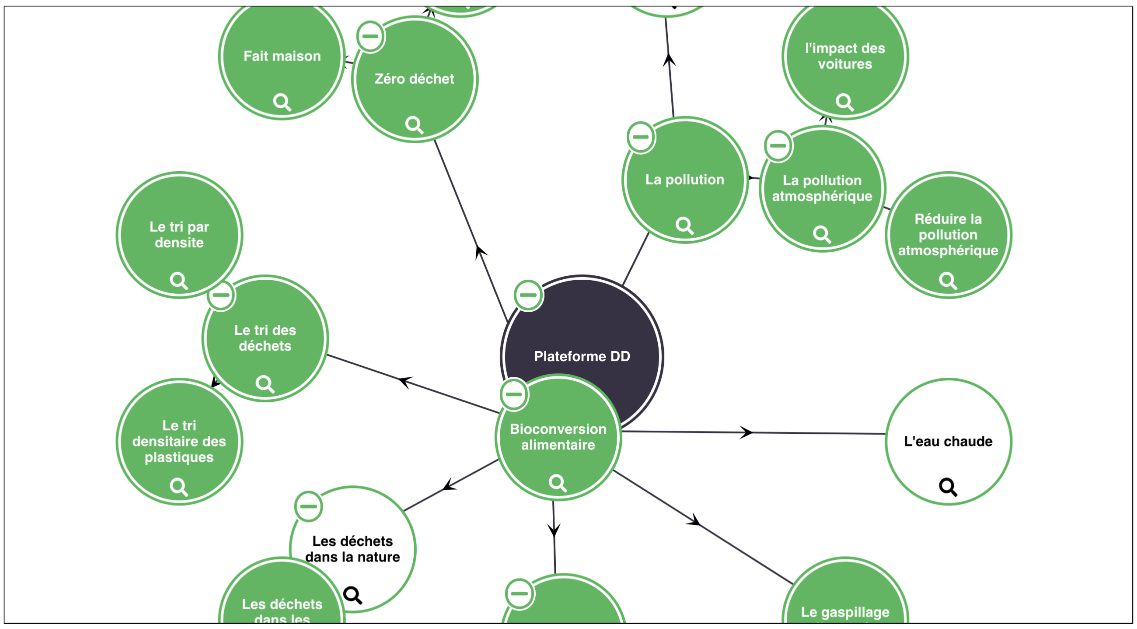


Figure 5.1: Plateforme DD Layout.

[1]https://sciences.brussels/dd/

5.1 Configuration

In Figure 4.3, we showed how we could alter the visualization of the data without affecting the default code. Remember that the props in that figure are not the only customizable props of the library. Figure 5.2 shows the differences in configuration when we compare the visualization of GuideaMaps with the one of Plateforme DD.

```
<ZoomableTree
    NodeComp={GMNode}
    LinkComp={GMLink}
    EditModalComp={
    ↪   GMEditModal}
    onAddNode={addGMNode}
    onDeleteNode={
    ↪   deleteGMNode}
    onNodeUpdate={args ->
    ↪   updateGMNode(args)}
    onVisibleChildrenUpdate={
    ↪   nodeId =>
    ↪   updateGMVisibleChildren(
    ↪   nodeId)}
/>
```

Listing 5.1: GM Configuration.

```
<ZoomableTree
    NodeComp={PDDNode}
    LinkComp={PDDLink}
    EditModalComp={
    ↪   PDDEditModal}
    onAddNode={() -> null}
    onDeleteNode={
    ↪   deleteGMNode}
    onNodeUpdate={args ->
    ↪   updateGMNode(args)}
    onVisibleChildrenUpdate={
    ↪   nodeId =>
    ↪   updateGMVisibleChildren(
    ↪   nodeId)}
/>
```

Listing 5.2: PDD Configuration.

Figure 5.2: The configuration differences between GuideaMaps and Plateforme DD.

5.1.1 NodeComp

The most obvious difference, which can immediately be seen when comparing both visualizations, is the layout of the nodes. In GuideaMaps (GM), we had rectangular nodes, while in Plateforme DD (PDD) the nodes are circular. To achieve this result, we implemented a special component, called *PDDNode*, to replace *GMNode*. From then on, every node in the data will be visualized by the code of PDDNode instead of the code of GMNode. Hence, GMNode still exists; it is not deleted or overwritten, but simply not used in this configuration.

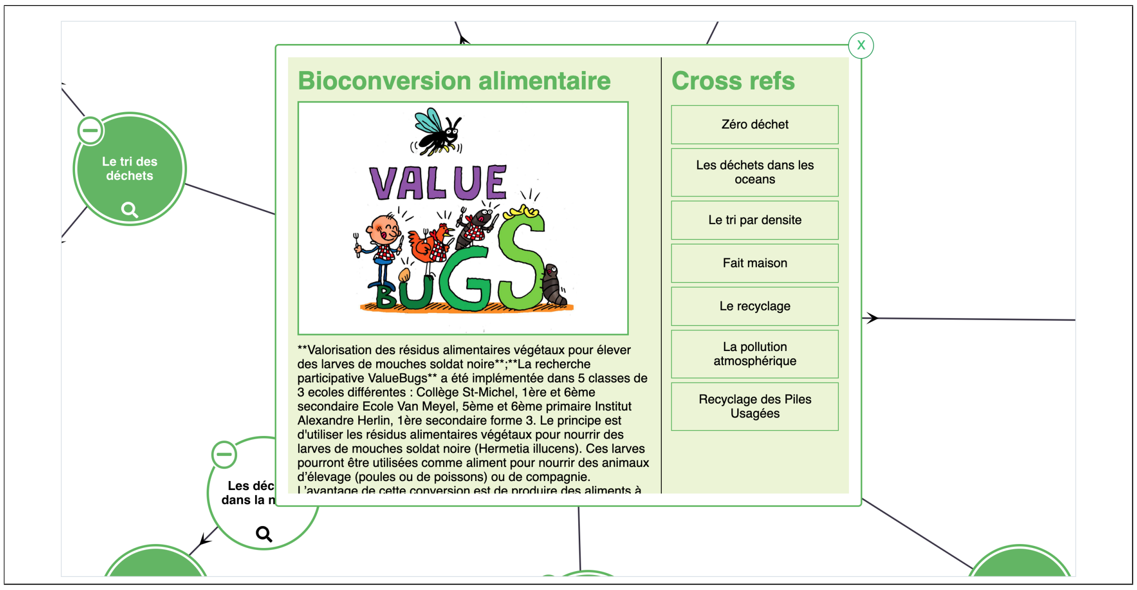
5.1.2 LinkComp

The second prop is responsible for the representation of the links. You will not observe big differences between a GMLink and a PDDLink, except for the thickness: a PDDLink is thicker than a GMNode. As a consequence, the implementation of PDDLink is a copy of the GMLink with the value for *strokeWidth* as only difference.

If the user does not need a different style for the links, he does not have to create a PDDLink component. In that case, it is possible to use GMLink as LinkComp, while the other props are eventually specially created for Plateforme DD.

5.1.3 EditModalComp

The edit modal of Plateforme DD is completely different compared to the one of GuideaMaps. First of all, it is not a real *edit*-modal because a user of the PDD visualization is not allowed to update the data of the nodes, he can only consult the available data, which is shown like in Figure 5.3.



Valorisation des résidus alimentaires végétaux pour élever des larves de mouches soldat noire;**La recherche participative ValueBugs** a été implémentée dans 5 classes de 3 ecoles différentes : Collège St-Michel, 1ère et 6ème secondaire Ecole Van Meyel, 5ème et 6ème primaire Institut Alexandre Herlin, 1ère secondaire forme 3. Le principe est d'utiliser les résidus alimentaires végétaux pour nourrir des larves de mouches soldat noire (Hermetia illucens). Ces larves pourront être utilisées comme aliment pour nourrir des animaux d'élevage (poules ou de poissons) ou de compagnie. L'avantage de cette conversion est de produire des aliments à

Figure 5.3: Plateforme DD Edit Modal.

At the left part of the edit modal, the actual content is shown. It starts with the title of the node, followed by a picture. If no picture exists, the text providing more information about the topic of the node is shown immediately under the title. Otherwise the title and the text are separated by the picture.

The size of the edit modal is fixed, but in case the information does not fit the modal, the user can scroll in the left part to continue reading until the end. The modal will not grow in size to make sure all data fits in it.

For some nodes, the edit modal will contain a list of so-called "cross references", which are links to nodes that in some way are related and which are not a direct child of the node. Such a link is not shown in the visualization, but by providing such a list, the user can still discover them. A click on a cross reference will close the modal and *move* the visualization until the node of the corresponding cross reference is centered. This allows the user to navigate to linked nodes without the need to show a spaghetti of links in the visualization.

5.1.4 Action Listeners

In GuideaMaps, we have a controls-div (section 4.4.3) containing a number of buttons to allow the user to perform some actions. In Plateforme DD, we chose for another approach. Because we have circular nodes instead of rectangular, it is less convenient to position multiple buttons next to each other at the bottom of the node. Further, we don't need a button to add child nodes because this is not allowed in Plateforme DD. Hence, we only have two buttons: one to expand and collapse the child nodes and one to open the edit modal.

The button to open the node is positioned in the center at the bottom of the node. A click on that button opens the modal and centers the node such that the node is in the middle of the screen when the user closes the modal. Specific to this implementation is that there is a second way to open the modal. If the user clicks once on the node, it is centered. If he clicks once more on the node (not necessarily on the button), the modal will open and the content will be shown. This is because a click on a centered node results in the same action as a click on the button to open the node.

The second button is one to expand or collapse the child nodes. As already said, we cannot position it next to the other button at the bottom of the node. Therefore we position it at the top-left of the boundary of the node. There, a circle is visible with a minus inside if the children are visible. A click on the minus will collapse the node such that the children are not visible anymore and replace the minus by a plus. When the user clicks on that plus again, the exact opposite action will happen: the node will be expanded and hence the children of the node will be visible again and the plus is replaced by a minus.

Note that a plus and minus can be used here as symbols to expand and collapse a node. This is possible because there can be no confusion with the action of adding a child node in Plateforme DD. If this would be possible, other symbols should be used to avoid such confusion and to not reduce the intuitiveness (Usability Requirement 1).

6 Evaluation

7 Conclusion & Future Work

7.1 Conclusion

7.2 Future Work

The presented application is not perfect yet and some improvements could be done in the future. One of the most important improvements that could be done is supporting a broader range of browsers. Currently, the tool works perfectly in the latest version of Google Chrome. It is good to have a working tool in one of the most widely used browsers, but lots of people use other browsers apart from Google Chrome. We detected some issues in Firefox, where for instance the tool is very slow on a tablet. In the future, we should improve the system such that it works in different browsers and if possible in all commonly used ones.

References

Balaid, A., Abd Rozan, M. Z., Norris Hikmi, S., & Memon, J. (2016). Knowledge maps: A systematic literature review and directions for future research. *International Journal of Information Management*, *36*, 451-475. doi: 10.1016/j.ijinfomgt.2016.02.005

Davies, M. (2011). Concept Mapping, Mind Mapping and Argument Mapping: What are the Differences and Do They Matter? *High Educ*, *62*, 279-301. doi: 10.1007/s10734-010-9387-6

De Troyer, O., & Janssens, E. (2014, August). A feature modeling approach for domain-specific requirement elicitation. In *Requirements Patterns (RePa), 2014 IEEE 4th International Workshop on* (p. 17-24). IEEE. doi: 10.1109/RePa.2014.6894839

Eppler, M. J. (2006). A comparison between concept maps, mind maps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing. *Information Visualization*, *5*(3), 202-210. Retrieved from `http://dblp.uni-trier.de/db/journals/ivs/ivs5.html#Eppler06`

ISO, E. (1998). *ISO 9241/11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) - Part 11: Guidance on Usability.*

Janssens, E. (2013). *Supporting requirement elicitation for serious games using a guided ideation tool on ipad.*

Moody, D. (2009, November). The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, *35*(6), 756–779. Retrieved from `https://doi.org/10.1109/TSE.2009.67` doi: 10.1109/TSE.2009.67

Moody, D. (2012). *The Physics of Notations: A Scientific Approach to Designing Visual Notations for Model Driven Development.*

O'Donnell, A. M., Dansereau, D. F., & Hall, R. (2002). Knowledge Maps as Scaffolds for Cognitive Processing. *Educational Psychology Review*, *14*(5). doi: 10.1023/A:1013132527007

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., & Carey, T. (1994). *Human-computer interaction.* Essex, UK, UK: Addison-Wesley Longman Ltd.

Wiegmann, D., F. Dansereau, D., C. McCagg, E., L. Rewey, K., & Pitre, U. (1992). Effects of Knowledge Map Characteristics on Information Processing. *Contemporary Educational Psychology*, *17*, 136-155. doi: 10.1016/0361-476X(92)90055-4