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

GUIDEAMAPS 2.0

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

2 Requirements

2.1 Functional requirements

The application should provide a lot of functionalities in order to be useful in many cases. The most important one is that the system should correctly show the current state of the data which is created by the user or loaded via a file. This data should be nicely visualized, easy to interpret and straightforward to edit. A difference with the first version of the GuideaMaps is that the tool should run in the browser. While version 1.0 was only designed for an iPad, a browser based version like GuideaMaps 2.0 immediately is a tool that can be used on multiple devices (e.g. tablets, laptops and desktops) and different operating systems. Hence, the user must not be limited to a particular device anymore, the tool can be used on any device with an internet connection. 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. The application will run on smartphones as well but it is not recommended to use it on devices with relatively small screens.

2.2 Usability requirements

In order to let a user create, explore and edit the data, the usability of the application should be as high as possible. This is an important requirement because the tool will not only be used by people with experience in Computer Science. It doesn't matter whether or not the user has a background in Computer Science, he should be able to easily learn the system in short time. Therefore, it is important to choose for clear, not misunderstandable icons on buttons where a click on this button invokes a certain action. Some examples of these actions are (1) adding a child node, (2) edit a node and (3) expand/collapse a node. Good icons for each of

the mentioned actions are shown in figure 2.1. The icons are not ambiguous, they can only be linked to one particular action and thus the user knows exactly what to expect when clicking on the button.

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

Figure 2.1: Good icons for the following actions: (a) add child node, (b) edit node, (c) expand node and (d) collapse node.

3
Related work

4 Implementation

4.1 ReactJS

ReactJS is an open source library to create user interfaces (*React – A JavaScript library for building user interfaces*, n.d.). One of its main goals is to provide the best possible rendering performances (Thinkwik, 2017). Performance is high 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 (*Why React? | React*, n.d.). Hence, this involves a huge benefit for the performance. Next to that, it is also not very hard to learn to code in React when 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 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.

4.2 d3

Another helpful tool is d3. With d3-hierarchy (*2D layout algorithms for visualizing hierarchical data.: d3/d3-hierarchy*, 2019), 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 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 places around it. Hence, the farther 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 won't be a spaghetti of links where you cannot see which node the link comes from and which node it is pointing to.

4.3 Architecture

With ReactJS and d3, the two most important pillars the application relies on are discussed. Now it is time to have a look at the architecture of the code, such that it is clear how all elements work together. Figure 4.1 shows a visualization of the structure of the code.

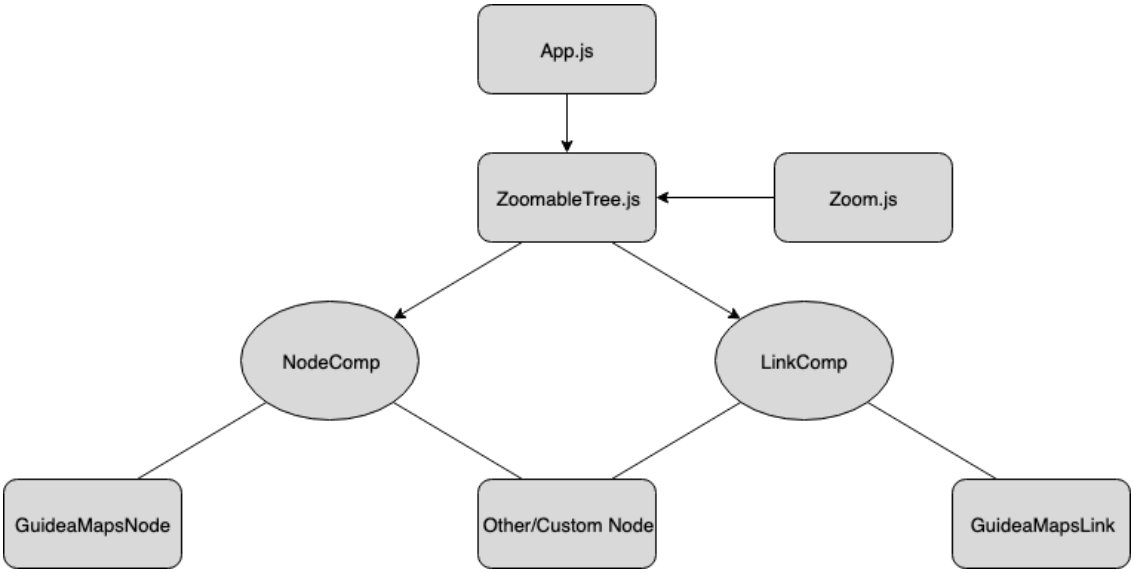

Figure 4.1: Architecture of the application.

The application is developed in such a way that it can be used as a library for other purposes than GuideaMaps as well. 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. When talking about the library, the layer of the NodeComp and LinkComp is very interesting and important. For the GuideaMaps, an implementation for a GuideaMapsNode and GuideaMapsLink is provided. Each implementation describes what every node and every link should look like in the visualization (TODO: provide more explanation about this somewhere else).

The strength of the library can be seen when a particular user would like to have a different representation for the nodes or the links or both. In that case, new components (e.g. MyCustomNode and MyCustomLink) should be implemented. In 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 GuideaMapsNode and GuideaMapsLink, respectively, by default. Hence, the only action that is required to *plug in* an other component is replacing GuideaMapsNode by MyCustomNode and GuideaMapsLink by MyCustomLink. Figure 4.2 shows the part of the code in App.js that should be adapted as explained. Note that NodeComp and LinkComp are not the only props that are passed to ZoomableTree. The others are omitted to improve the readability.

```
<ZoomableTree
    NodeComp={GuideaMapsNode}
    LinkComp={GuideaMapsLink}
/>
```

Listing 4.1: Default components.

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

Listing 4.2: Custom components.

Figure 4.2: Two listings showing how to use the library.

4.4 Styling

The styling of the application is very important for the end user. Everything should look pretty and as already mentioned in section 2.2, the possibilities should be straightforward and visible. Therefore, the styling has to be as good as possible. While developing and creating a beautiful style for applications and websites, the code for these styles can become a big part of the implementation. Hence, a good framework is necessary to reduce the lines of code to a reasonable number.

Tailwind CSS (*Tailwind CSS - A Utility-First CSS Framework for Rapid UI Development*, n.d.) is a framework that helps developers with styling. 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 don't always wanted and it can be quite hard to undo the part you didn't ask for. With tailwind on the other hand, you can grab only the features you want, without side-effects. Figure 4.3 shows an example with two small listings. The first uses inline style while the second makes use of tailwindCSS.

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

Listing 4.3: Normal CSS, no tailwind.

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

Listing 4.4: With tailwind CSS.

Figure 4.3: Difference when using tailwind CSS or not.

5 Conclusion

5.1 Future work

Describe future work here.

References

2d layout algorithms for visualizing hierarchical data.: d3/d3-hierarchy. (2019, February). D3. Retrieved 2019-02-25, from https://github.com/d3/d3-hierarchy (original-date: 2015-07-15T18:01:35Z)

React – A JavaScript library for building user interfaces. (n.d.). Retrieved 2019-02-13, from https://reactjs.org/index.html

Tailwind CSS - A Utility-First CSS Framework for Rapid UI Development. (n.d.). Retrieved 2019-02-25, from https://tailwindcss.com/

Thinkwik. (2017, December). *Why ReactJS is gaining so much popularity these days.* Retrieved 2019-02-13, from https://medium.com/@thinkwik/why-reactjs-is-gaining-so-much-popularity-these-days-c3aa686ec0b3

Why React? | React. (n.d.). Retrieved 2019-02-13, from https://facebook.github.io/react/docs/why-react.html