Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Cybernetics

Intelligent tree select component

Project report and description.

Lečbych Jakub

Bc. programme: Software engineering and technology

Branch of study: Programmer/ Architect of web applications

Supervisor: Ing. Petr Křemen, Ph.D.

Prague, January 2018

Acknowledgements

I would like to thanks to my supervisor Ing. Petr Křemen, Ph.D. for his patience, advice, and feedback during my work on this project. Also I thanks for providing materials like example data and useful web sources to this topic.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

1. Background

1.1 Linked Data

What are linked data? (Bizare, Heath, & Berners-Lee) The concept of linked data is simple. You don't need any experiences with web programming to understand what Link Data are. Firstly, lest start what data is. Data is everything that can be measured, analyzed, collected, and visualized. They are sets of values consisting of pieces of information. So data, for example, are images, videos, documents, web pages, and so on.

Now with all that data, you can put it on the internet. If u have some web page, for example, Wikipedia page, this page has some data in it, it also has some links to another page where you can learn more. For humans, it's easy to understand what is on the page. You can recognize the image, links, videos and so on. But for computer, it's hard to understand what on the page is. So this is basically the concept of linked data. You take the data from your page, package them up in the way that the express what data it is, so it can be easily understandable for computer and humans.

With all of this, there are two main problems. First is what format you use for your data. There are lots of formats, for example JSON, RDFa, XML, CSV, HTML, etc. And the second problem is how we can link the pieces of data together. Most common and easiest way to express this data is on the key-value property.

1.2 RDF

Resource Description Framework (RDF) (Klyne & Carroll) (Černá, 2013)is a general description framework for describing web sources. It is a basis for the semantic web. RDF is a general frame for the description, exchange, and reuse of metadata, it assigns a semantic to web sources. RDF can be represented as a graph or triplet.

For graphs – subject and object are nodes and predicates are edges. On the other hand, triplets are described as – source, property, and value. Triplet in official terminology express some facts about the source. A claim consists of three pieces that together create a sentence: subject → predicate → object. Within this statement, the source is a subject identified by URI (or IRI), the property is a predicate (what we say about the source) and value is an object. Predicates that we used for describing a source comes from so-called schemas – that are vocabularies or ontologies. Examples can be Dublin Core (DC) or Friend of a Friend (foaf) metadata standards.

RDF syntax has various type of formats that are called serialization formats. Among these formats are for example Turtle, N-Quads, N-Triplets, and JSON-LD.

1.3 JSON-LD

JSON-LD is an RDF syntax for describing linked data using JSON format. (Sporny, Longley, Kellogg, Lanthaler, & Lindström, 2014) JSON-LD is both JSON document and RDF document, but it has some differences with RDF. First, JSON-LD properties can be URIs (or IRIs) or blank nodes whereas in RDF properties must be URIs (or IRIs). This means that RDF datasets can be serialized by JSON-LD. On the contrary, it is not possible. Second, JSON-LD object lists are part of data model whereas RDF objects are part of the vocabulary. And the last one, RDF values are either literals or language-tagged strings whereas JSON-LD also supports JavaScript native types, that are numbers, booleans, and strings.

EXAMPLE 2: Sample JSON-LD document using full IRIs instead of terms

```
{
  "http://schema.org/name": "Manu Sporny",
  "http://schema.org/url": { "@id": "http://manu.sporny.org/" },  ← The '@id' keyword means 'This value is an identifier that is an IRI'
  "http://schema.org/image": { "@id": "http://manu.sporny.org/images/manu.png" }
}
```

Figure 1 Example of JSON-ld document

1.4. React and Redux

React is JavaScript UI framework developed and maintained by Facebook. Its one of the most used framework because it's easy to learn and because of its performance efficiency. If you are familiar with MVC model, React is only the view part. More about React you can learn here https://reactjs.org/.

Redux is completely independent on React. Redux is a state container for JavaScript. This means, that Redux is just a framework for managing the state of your web applications. It evolves from Flux framework but does not take Flux complexity. More about Redux you can learn here: https://redux.js.org/.

2. Introduction

2.1 Overview

As developers, you could ever need a select input that can render options as a tree. There are plenty of components, even default HTML input element that enables you to render a drop-down box with options, but almost none of them enables you to render these options as a tree. And if you finally find some, there was no way to customize the visual site of this component or this component can't filter fast enough between options. So, you in many cases ended up creating your own custom implementation, that was used for that one particular case. This was the reason why I have created Intelligent Tree Select component which retains

simplicity but also flexibility, search speed, reusability, and mostly it can render options in both ways – normally or as a tree. It can perform well even with a large list of options dues to the intelligent way of rendering these options. And everything can be customized for you so it can perfectly fit your website.

2.2 Scope of research

Data, pieces of information that are measured, collected, analyzed, and used for a variety of things. This data can be simple or structured, specific or indefinite etc. Structured data are usually linked with another data and can be represented as a graph. For humans, it's easy to understand the connection between this data, but the computer does not have this kind of intuition. Let's imagine that you are on Wikipedia web page, there is usually some images, references, and links to another web pages. As I said earlier we don't have a problem to understand what is in that image or what information is behind that link. But the computer, see that there are only some images or links. And that's why the concept of Linked Data was created. It allows the computer to understand the connection between data. So, if you ask the computer for example "Who is the president of USA?" the computer can go to the Wikipedia, find a page about the USA then follow the link to the president and tell you the answer.

Data are in many cases provided through the Web APIs. Web APIs are defined as an interface through which your application can communicate with server typically via HTTP request messages that also define what kind of response data you want. Usually, response message contains data in XML or JSON format. Sometimes data are provided as plain text, CSV, etc. Next, think you can define is the maximum length of data you want to receive or any other conditions. This information is not defined in HTTP request header but usually in URL you are requesting. Some APIs provides an interface that enables you to filter on the server side, which is usually faster than filtering on the client side.

With all these data, there is a question – "How to efficiently render these data as options?" – because if you are not familiar with rendering large lists. This operation can be really time-consuming and can affect the performance of your application significantly. And as developers you probably know, that performance is one of the most important aspects of applications.

2.3 Outline

The first chapter will cover the component itself. At the beginning, we will look at the technical parts of the component such as class diagram and sequence diagram. Then the main part of Intelligent Tree Select component will be described. And at the end of this chapter, we will look at other parts of the component, such as a modal form for creating new options or settings that enable customization of the select component. The second chapter is about APIs

of the component and how developers can customize rendering or filtering the options. In the third chapter, we will look at the Linked Data. How they are described and in their syntax. Also, there will be shortly described the most common format – JSON-LD document for Linked Data that is used for describing Linked Data and that is readable for humans and computers.

CHAPTER 2

3 Intelligent Tree Select component

Intelligent Tree Select component is a custom JavaScript component build with React framework and Redux framework. This component enables the user to filter among multiple options, display them as a tree or create a new option. This component is a wrapper around another component that is called VirtualizedTreeSelect component. The VirtualizedTreeSelect component is the core component that performs filtering, selecting, and displaying all the options. The other component, Intelligent Tree Select component, expand the functionality of the VitualizedTreeSelect component.

3.1 Sequence diagram

On the image below you can see sequence diagram of the Intelligent Tree Select component.

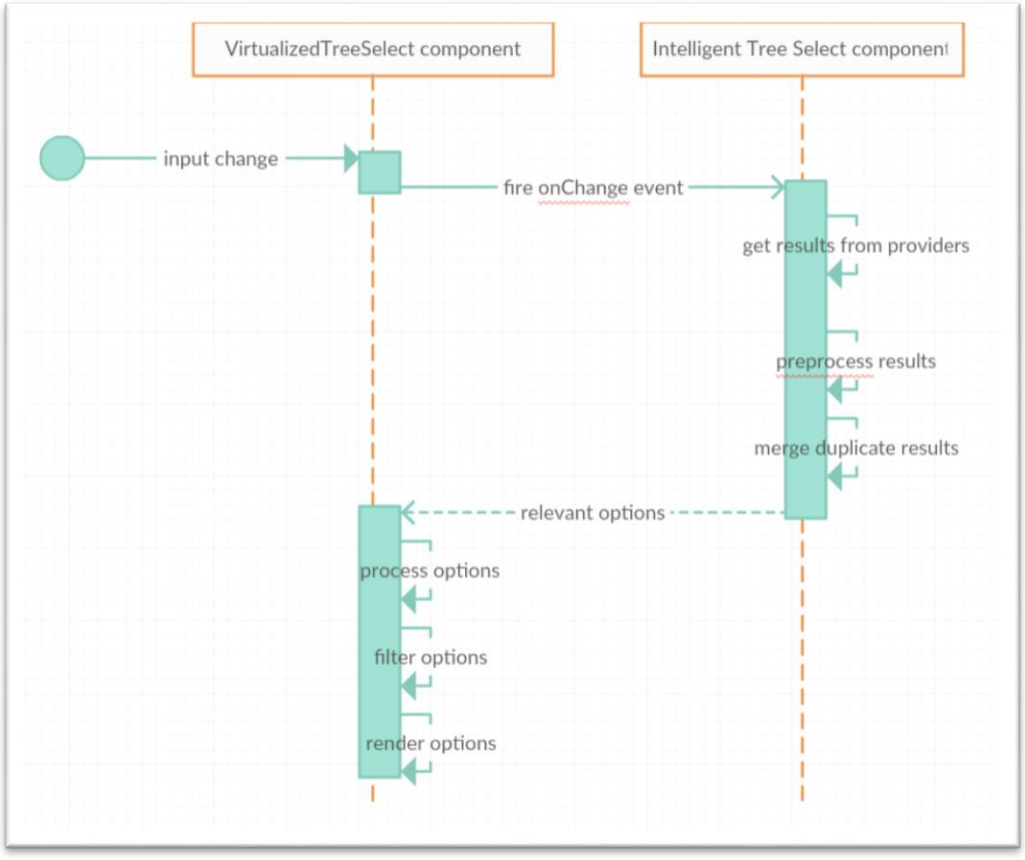

Figure 2 Sequence diagram of options filtering and rendering

As you can see on this diagram when user click, type or paste into the select input field, the VirtualizedTreeSelect (VTS) component fire an event to which the Intelligent Tree Select (ITS) component listen. When this event occurs, the ITS component fetches the responses from the providers, and preprocess them. This means it adds a custom property such as 'provider' and 'state'. Then options are merged according to the priority of the providers. Because two different providers could return same or slightly different results. After all of that, all options are passed back to the VTS component. VTS component then sorts then process this options again, in this case, it adds values such as a pointer to the parent, depth, and graph properties. Then Options are filtered and filtered options are rendered. This process of filtering and rendering will be described in detail later.

3.2 Class diagram

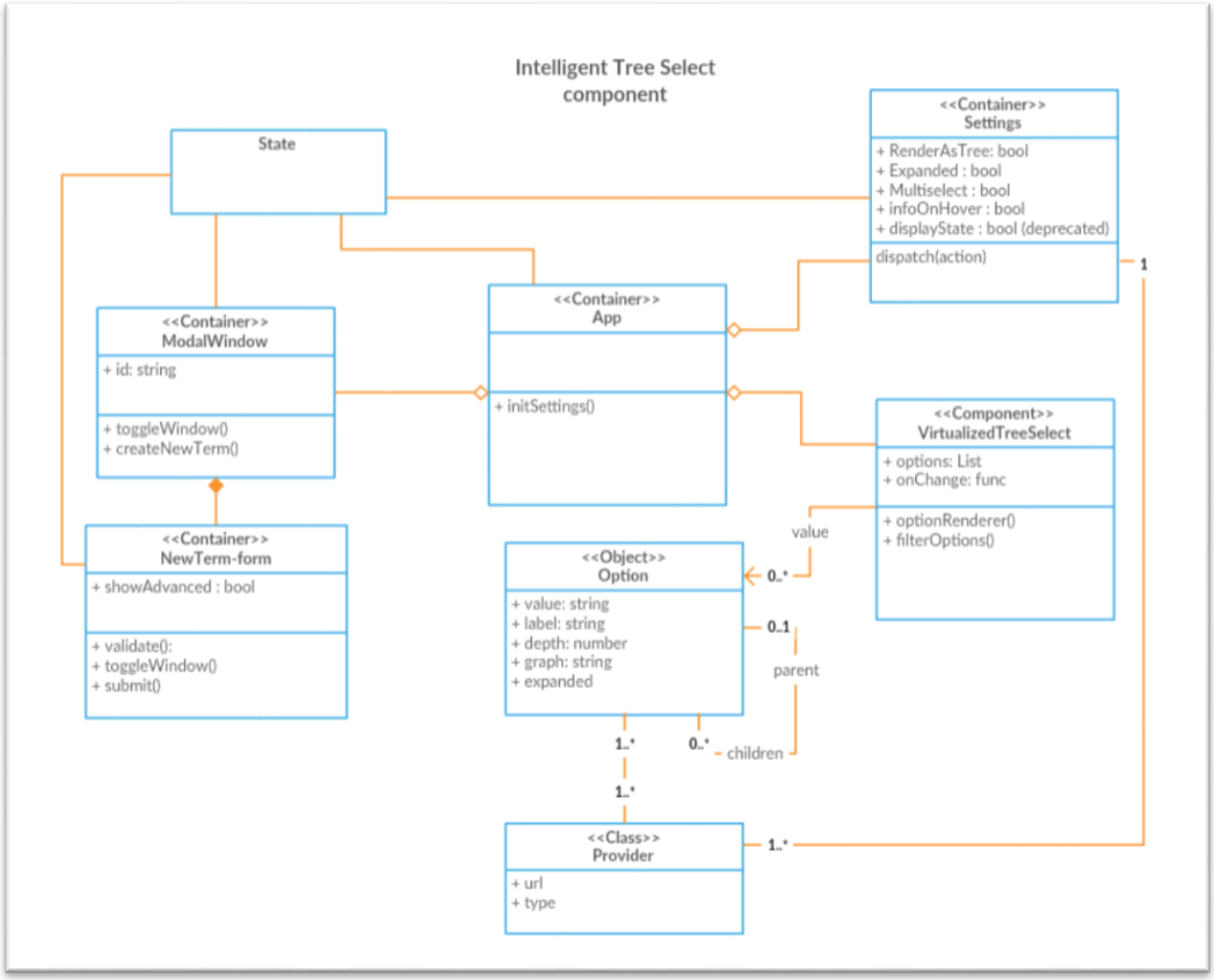

Figure 3 Class diagram of Intelligent Tree Select component

As you can see in the class diagram, there is one key class called App, this class just glue up every other class together. This is also the main class that is being injected into the website. In this class in injected other classes such as ModalWindow, this class represents a button with 'onClick' event, that renders modal dialog only with a header. Into this modal dialog is injected other class that represents the modal body and modal footer with redux-form for creating new options and actions buttons for submitting and closing this form and modal dialog. Another class is Settings, this class represents the collapsible HTML element with a form that has some checkboxes for changing the settings of the main component called VirtualizedTreeSelect. This class it the core class that renders the select input with a select box and perform actions such as filtering and rendering options as a tree, selecting or multi-selecting, etc.

VirtualizedTreeSelect component accepting several attributes, these will be described in API capitol. One of the acceptable attributes is options. Options should be provided as an array of JavaScript objects. Each object represents one node of the tree. The node must have three properties – label (representing visible string), value (unique string or numeric identification of that node) and children (string or array of strings that representing edge/s between nodes). Nodes can also contain other properties like disabled, this property indicates if the option is disabled or not.

```
let options = [
    {
        label: 'options 1',
        value: '1',
        children: ['2', '3']
    },
    {
        label: 'options 2',
        value: '2',
        children: ['4']
    },
    {
        label: 'options 3',
        value: '3',
        children: [ ]
    },
    {
        label: 'options 4',
        value: '4',
        children: [ ]
    },
    ]
```

Figure 4 Example of input array representing options

3.3 VirtualizedTreeSelect part

Main part 'VirtualizedTreeSelect' component is custom component build on 'react-virtualized-select' [1] and 'react-select' [2]. This component retains the same API as both components, in addition to it provide several new configurations, that will be described below. So as React-Select, this component generates hidden text input field that contains the value of the selected option, so it could be submitted as part of the standard form.

When the option is selected, 'onChange' event is fired and this event return selected option. All the changes to the select input must be handled by the user; the user must pass that event value to the 'value' attribute of the select component.

3.4 Modal redux-form part

This component part consists of two dependent react classes. The first one renders empty modal dialog, that contains an only header and close button. The second one renders the actual redux-form in modal body and actions buttons for submitting or canceling in the modal footer.

As I mentioned earlier, this redux-form is used for creating new Nodes. It has several form fields:

- Option label (required) – representing value that is visible in drop-down box
- Option value (required) – representing unique ID
- Option description -
- Children – multi-select box containing labels of all other options
- Parent – select box containing labels of all other options
- Etc.

After each key press validation is triggered so the user is informed about invalid inputs before submitting that form. Also, the form is submitted only when all fields are valid. After that new node is created and its added to current tree graph and event 'onNewOptionCreation' [3] will be fired.

[1] https://github.com/bvaughn/react-virtualized-select
[2] https://github.com/JedWatson/react-select
[3] This part is not implemented yet

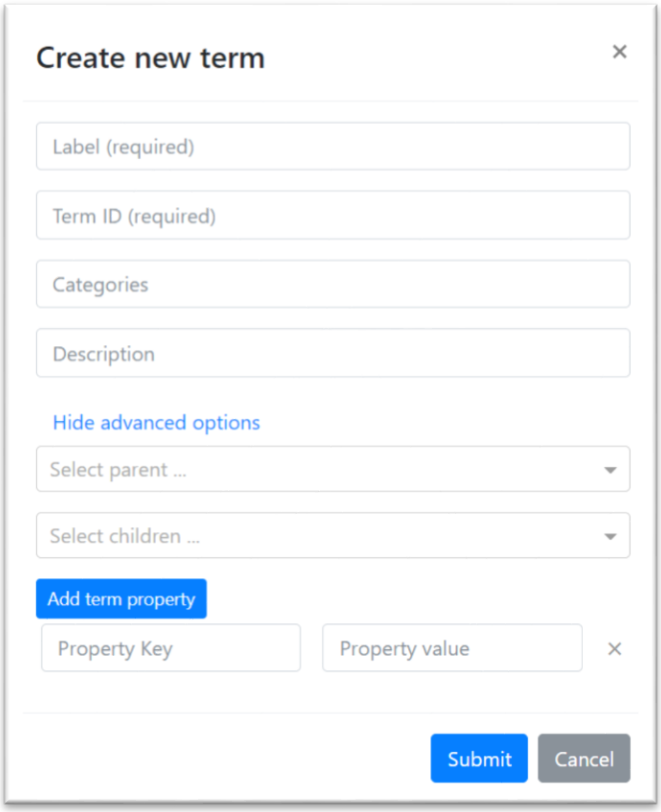

Figure 5 Redux-form for creating new options (terms)

3.5 Settings part

This is just a collapsible form with several checkboxes that provide some changes to the 'VirtualizedTreeSelect' component, like expand/ collapse all. Multi-select, this option, if it is checked then the component will provide multi-selection otherwise only one option will be selectable. Render as a tree, as the name suggests, this option renders all nodes as a tree also it slightly change filtering because by default if this option is checked the filtering will also show the whole path in the tree, meaning all parents until root parent will be displayed as well. Display info on hover, this option enables to show additional information for that node on hover. For example, description.

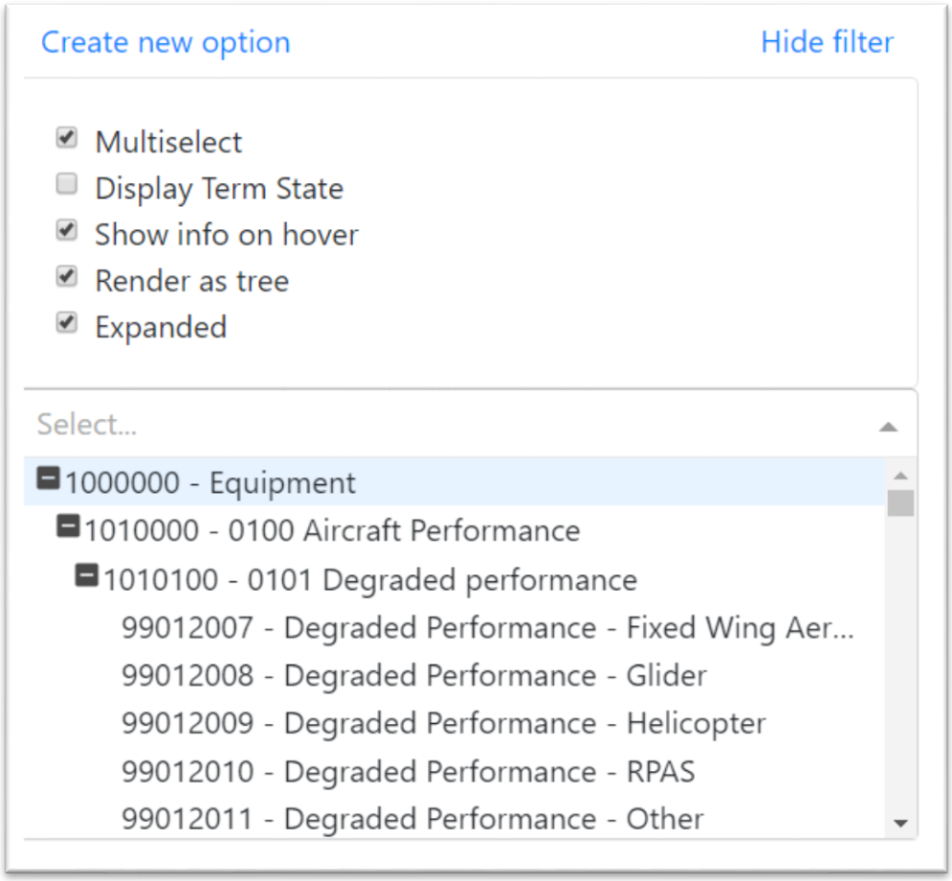

Figure 6 Intelligent Tree Select component with visible dropdown and settings part

3.6 VirtualSelectTree Component life cycle

Now let's deeper look into the main component life cycle. First time after the component is created, the options provided to this component via props will be processed, this means that every option will get component custom properties – parent (string pointer to the parent node), depth (numeric value depth of the node in tree), graph (string value representing position of the node in tree), and expanded (Boolean value representing if children will be visible or not). Example of graph property is: "0-1-0", this means that node with graph property of that value is the first node with depth 2 whose parent is the second node with depth 1 whose parent is the first node with depth 0.

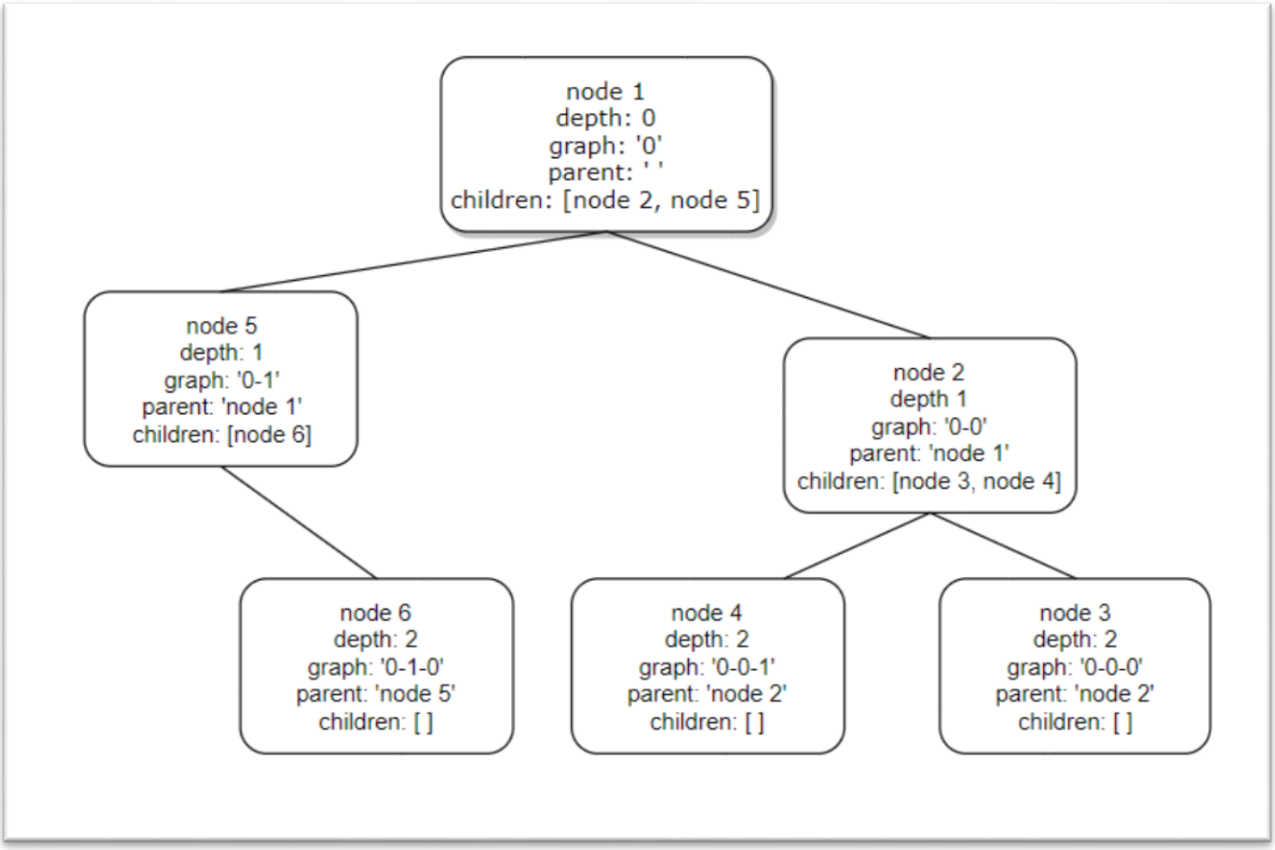

Figure 7 Visual example of tree graph representation

These three custom properties are not really necessary for the correct representation of tree but they help in faster filtering and correct visual representation of the tree.

During this processing, all options are sorted in way of depth-first. See below.

```
var options = [
    {graph: '0', ...},
    {graph: '0-0', ...},
    {graph: '0-0-0', ...},
    {graph: '0-0-1', ...},
    {graph: '0-1', ...},
    {graph: '0-1-0', ...},
    {graph: '1', ...}
    ...
    ]
```

Figure 8 Example of sorted array representing tree data

Sorting is another really big problem. Length of the options can be more than thousands so choosing right sorting algorithm is tricky. To tell truth this is the area where I still need to work on. One way should be using a different data structure for options such as HashMap. But this will just solve one part of many. Then this new processed sorted array is passed into the component itself as options prop. After all of that, 'render' method is called and the component is properly rendered.

Because rendering is really slow process compared to all JavaScript calculation, component renders only necessary HTML elements. Firstly, select input is rendered, nothing more, nothing less. Dropdown menu with options is rendered only when select input element gets focus. Speaking of options, tree data could have thousands of options and render all options at once is not really smart idea and this is where this component shines. Time needed for rendering 20 options is most likely same as the time needed for rendering 100 options and that is because during the options rendering process component remember the index of currently focused option and render just a few options before and after that index.

Filtering of options is fired after each key press. Firstly, all options whose label value include search string are added to result's list then all parents for these result options are as well then this result's list is returned to the component and component render appropriate options.

CHAPTER 3

4 API

All available select props are described here: https://github.com/JedWatson/react-select#select-props and here: https://github.com/bvaughn/react-virtualized-select/#react-virtualized-select-props. Additional parameters used by VirtualizedTreeSelect component are described in this table:

Property	Type	Default value	Description
childKey	PropTypes.string	'children'	Attribute of option that contains the values of children options
expanded	PropTypes.bool	True	Attribute if all options are by default expanded or not
renderAsTree	PropTypes.bool	True	Attribute if options should be rendered as a tree. If false options are rendered normally as for default select
optionRenderer	PropTypes.func		Custom way to render options (see below)
filterOptions	PropTypes.func		Custom way to filter options (see below)

Table 1 Custom attributes supported by VirtualizedTreeSelect component

4.1. Custom option renderer

This prop is useful if u want to override the default optionRenderer method

Property	Type	Description
focusedOption	`PropTypes.object`	The option currently-focused in the drop-down. Use this property to determine if your rendered option should be highlighted or styled differently.

focusedOptionIndex	`PropTypes.number`	Index of the currently-focused option.
focusOption	`PropTypes.func`	Callback to update the focused option; for example, you may want to call this function on mouse-over.
labelKey	`PropTypes.string`	The attribute of option that contains the display text.
option	`PropTypes.object`	The option to be rendered.
options	`PropTypes.arrayOf (PropTypes.object)`	Array of options (objects) contained in the select menu.
selectValue	`PropTypes.func`	Callback to update the selected values; for example, you may want to call this function on click.
style	`PropTypes.object`	Styles that must be passed to the rendered option. These styles are specifying the position of each option (required for correct option displaying in the drop-down).
valueArray	`PropTypes.arrayOf (PropTypes.object)`	An array of the currently-selected options. Use this property to determine if your rendered option should be highlighted or styled differently.

valueKey	`PropTypes.string`	Attribute of option that contains the value.
onToggleClick	`PropTypes.func`	Callback to event for clicking on expand button
childrenKey	`PropTypes.string`	Attribute of option that contains the values of children options

Table 2 Attributes required for custom renderMethod

4.2. Custom filter options

By default, a component uses a custom function for filtering the options. This function is built on react-select-fast-filter-options[4] . I don't recommend overriding this method unless you know what you are doing. For more details, you can look at https://github.com/JedWatson/react-select#advanced-filters

[4] https://github.com/bvaughn/react-select-fast-filter-options

CHAPTER 4

5. Future work

In this report we have described the technical parts of the component, examine the concepts of Linked Data, RDF, semantic web, and JSON-ld, and we have come up with problems that need to be solved.

- The biggest one is performance and how we can efficiently process and sort an array of options.
- Next one is filtering. Currently, filter method runs on the whole array of options, but we don't really need to iterate over each option. We can take the bottom-up approach in filtering. This means start at the end of each tree branch and move up to the root until we came to the root (this is the worst case scenario) or until we find some node satisfying the condition of the filter, then we can safely put the whole tree branch from the root to this node.
- The third problem is how we can efficiently get data from different providers, merge them and pass the result into the VirtualizedTreeSelect component.

Also, we need to implement the support for different options providers. This version of Intelligent Tree Select component only supports options directly pass into it. With this, we need to figure out how to convert responses to a unified format. Some providers can support for example only XML or CSV format, but others can support for example only XML or JSON.

After that, we can get into benchmarking and comparing this solution with other solutions. So these are the key issues that need to be solved in the future.

6. References

Bizare, C., Heath, T., & Berners-Lee, T. (n.d.). *Linked Data - The Story So Far.* Retrieved from http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf

Černá, A. (2013). *Sémantický popis UML modelů pomocí OWL ontologií.* Brno.

Klyne, G., & Carroll, J. J. (n.d.). *Resource Description Framework (RDF): Concepts and Abstract Syntax.* Retrieved from W3C: https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., & Lindström, N. (2014, Jan 16). *JSON-LD 1.0.* Retrieved from W3C: https://www.w3.org/TR/json-ld/