**Czech technical university in Prague**

**Faculty of Electrical Engineering**

**Department of Cybernetic**

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*

Table of Contents

[Intelligent tree select component 3](#_Toc502781583)

[Modal redux-form part 3](#_Toc502781584)

[Settings part 3](#_Toc502781585)

[VirtualizedTreeSelect part 4](#_Toc502781586)

[VirtualSelectTree Component life cycle 4](#_Toc502781587)

[API 6](#_Toc502781588)

[Custom option renderer 6](#_Toc502781589)

[Custom filter options 8](#_Toc502781590)

[RDF 9](#_Toc502781591)

[JSON-LD 9](#_Toc502781592)

# Intelligent tree select component

It is a custom react-redux component that enables the user to filter among multiple options that are represented as a tree. 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.

This component consists of three independent parts. First and the main part is “VirtualizedTreeSelect” component. The second one is a modal window with redux-form for creating new options (nodes). The last one is settings that enable the user to switch between different modes, for example, “render options as a tree”, “expand/collapse all” and so on.

## Modal redux-form part

This component part consists of two dependent react classes. The first one renders empty modal dialog, that contains 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. Two required other optional. The required fields are text field whose value will be Node label and second required field is text field whose value will be Node value (the one representing ID). Other optional fields are for example select for a parent, multi-select for children, text input for description and so on.

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’ [[1]](#footnote-1) will be fired.

TODO example

## 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.

## VirtualizedTreeSelect part

Main part ‘VirtualizedTreeSelect’ component is custom component build on ‘react-virtualized-select’ [[2]](#footnote-2) and ‘react-select’ [[3]](#footnote-3). 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 of the select input must be handled by the user; the user must pass that event value to the ‘value’ attribute of the select component.

TODO example

## VirtualSelectTree 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) and graph (string value representing position of the node in tree). 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.

TODO visual example

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.

e.g.

options = [

{graph: ‘0’, …},

{graph: ‘0-0’, …},

{graph: ‘0-0-0’, …},

{graph: ‘0-0-1’, …},

{graph: ‘0-1’, …},

{graph: ‘0-1-0’, …},

{graph: ‘1’, …},

…

]

TODO to img

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 rendering 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.

## 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 three. 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 filer options (see below) |

### 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 |

### 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]](#footnote-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

# RDF

Resource Description Framework (RDF) 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.

# JSON-LD

JSON-LD is an RDF syntax for describing linked data using JSON format. 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.

[](https://www.w3.org/TR/json-ld/)

Figure 1 example of JSON-ld document

1. This part is not implemented yet [↑](#footnote-ref-1)
2. https://github.com/bvaughn/react-virtualized-select [↑](#footnote-ref-2)
3. <https://github.com/JedWatson/react-select> [↑](#footnote-ref-3)
4. https://github.com/bvaughn/react-select-fast-filter-options [↑](#footnote-ref-4)