# Intelligent tree select component

It is custom react-redux component that enable user to filter among multiple options that are represented as tree. Options should be provided as array of JavaScript objects. Each object represents one node of tree. Node must have three properties – label (representing visible string), value (unique string or numeric identificator 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 option is disabled or not.

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

## VirtualizedTreeSelect part

Main part ‘VirtualizedTreeSelect’ component is custom component build on ‘react-virtualized-select’ [[1]](#footnote-1) and ‘react-select’ [[2]](#footnote-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 value of selected option, so it could be submitted as part of standard form.

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

TODO example

## Modal redux-form part

This component part consists of two dependent react classes. First one render empty modal dialog, that contains only header and close button. Second one render the actual redux-form in modal body and actions buttons for submitting or canceling in 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 parent, multi-select for children, text input for description and so on.

After each key press validation is triggered so user is informed about invalid inputs before submitting that form. Also 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]](#footnote-3) will be fired.

TODO example

## Settings part

This is just 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 component will provide multi-selection otherwise only one option will be selectable. Render as tree, as the name suggest, this option render all nodes as tree also it slightly change filtering because by default if this option is checked the filtering will also show 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.

## 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 first node with depth 2 whose parent is second node with depth 1 whose parent is first node with depth 0.

TODO visual example

These three custom properties are not really necessary for 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 bellow.

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.

# RDF

Resource Description Framework (RDF) is a general description framework for describing web sources. It is a basis for semantic web. RDF is a general frame for the description, exchange and reuse of metadata, it assigns a semantic to a 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. Claim consist of three pieces that together create a sentence: subject 🡪 predicate 🡪 object. Within this statement, the source is a subject identified by URI (or IRI), property is a predicate (what we say about the source) and value is a 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 have 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 a RDF syntax for describing linked data using JSON format. JSON-LD is both JSON document and RDF document, but it have 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 vocabulary. And 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

[Figure 1 example of json-ld document 1](#_Toc501975109)

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