One quick look at how to create a Redux-connected React component / feature

2019-10-23 version by JV – **UNDER REFINING – Though possibly never final.**

/actions/ActionTypes.js: Defining three new allowed action types / signal types / message types

- _REQ = (AJAX) requested _OK = success back _X = failure back. These are not any standards, just Juhani's convention. Short, easy to spot in the Redux Dev Tool

```
CATEGORIES_SEARCH_REQ:   'CATEGORIES_SEARCH_REQ',
CATEGORIES_SEARCH_OK:    'CATEGORIES_SEARCH_OK',
CATEGORIES_SEARCH_X:     'CATEGORIES_SEARCH_X',
```

/actions/category.js: Defining three new allowed ways (only ways) to create action objects / signals, signal objects / messages, message objects

```
// ACTION CREATORS (Action object creator functions)

// CATEGORIES SEARCH BY KEYWORD
export const categoriesSearch_REQ = (keyword) => ({
  type: ActionTypes.CATEGORIES_SEARCH_REQ,
  keyword: keyword,
});
export const categoriesSearch_OK = (categoryList) => ({
  type: ActionTypes.CATEGORIES_SEARCH_OK,
  categoryList: categoryList
});
export const categoriesSearch_X = () => ({
  type: ActionTypes.CATEGORIES_SEARCH_X,
});
```

/actions/category.js: Defining one Redux “service” (function) that redux-connected React components can use to make an intelligent search (from backend DB to the frontend Redux store)

- Exported for other modules to use
- Asynchronous as React <-> Redux connection is asynchronous
- We signal what we are doing, and do it (_REQ + axios request). In the case of Redux action the dispatch both signals and does it (_OK(payload) /_X).

```
export function categoriesSearchByKeyword(keyword) {

  return async (dispatch, getState) => {

    dispatch(categoriesSearch_REQ(keyword));

    const ajaxRequest = {
      method: 'get',
      url: API_ROOT + `/category/search/${keyword}`,
    };

    axios(ajaxRequest)
      .then((response) => {
        dispatch(categoriesSearch_OK(response.data));
      })
      .catch((error) => {
        console.error("Error: " + error);
        dispatch(categoriesSearch_X());
      })
      .then(() => {
        return {
          type: null
        }; // 'Empty' action object
      });
  }
};
```

- (((At the end the empty action that is returned will be handled by the reducer, but as the type is null, no action is taken. Without this, we might get error messages to console. When will that empty dummy Action object be dispatched and handled? Well, look later when the React component will dispatch a call to this categoriesSearchByKeyword.)))

/reducers/category.js: We add a new category list for the search results to the redux store initial state. (((Later we could make this more elegant and just use the list of found id:s (if our categoryList has all Categories from database))))

```
import ActionTypes from '../actions/ActionTypes';

// Define initial states of reducer
export const initialState = {
    isLoading: false,
    categoryList: [],
    categorySearchList: [],     <= just this place/container here was added to Redux store
    categoryIdsFound: null,
    categoryCurrent: null,
};
```

/reducers/category.js: We add there new handlers to the **categories** reducer. (The name of the reducer (function) will be the name of the subarea/object in the Redux store state: *state.categories.categorySearchList.*

- These cases tell what the reducer should do when it receives any of these three actions. Like the picture states, reducer gets the old state and new action, and returns the new state
- Redux puts that new state to the store auto-magically (that code not visible to use, we just see in /src/index.js that the store and the reducers are connected).

```
export default function categories(state = initialState, action) {

   switch (action.type) {

       ... other cases...
       case ActionTypes.CATEGORIES_SEARCH_REQ:
        return {
            ...state,
            isLoading: true,
        };
       case ActionTypes.CATEGORIES_SEARCH_OK:
        return {
            ...state,
            categorySearchList: action.categoryList,
            isLoading: false,
        };
       case ActionTypes.CATEGORIES_SEARCH_X:
        return {
            ...state,
            isLoading: false,
        };
        ... other cases...
```

/components/categoryComponents/CategoriesIntelligentSearch.jsx: Simples possible, yet enlightening example of a redux-connected React component. Both data-bound (data refreshed automatically from Redux store) and action-bound (events dispatched to the Redux code.

- First the component looks like any other React component. It just strangely relies on props that are not given by the parent component, but appear to fall from the sky.

```
import React, { Component } from 'react';
import CategoryItem from './CategoryItem';

// One of the simplest possible sensible Redux-connected React examples:

class CategoriesIntellingentSearch extends Component {

  componentDidMount() {
    let keyword = "Training";
    this.props.categoriesSearchByKeywordLocal(keyword);
  }

  render() {
    let categorySearchList = this.props.categories.categorySearchList;
    return (
      <div>
        <h2>Found Categories based on keyword</h2>
        <p>
        {
          categorySearchList.map(
            (item, index) => (<CategoryItem item={item} />)
          )
        }
        </p>
      </div>
    );
  }
}
```

- The `CategoryItem` child component is a **so-called presentational component**. It merely presents=shows the data the parent feeds it via its props, here the prop 'item'
- The `CategoriesIntellingentSearch` on the other hand is called a connected component.

/views/categoryViews/Categories.jsx: Simply uses the CategoriesIntelligentSearch component, but does not provide any props to it! Mystery about those props intensifies!

```
        <h2>My Categories</h2>
        <CategoriesIntellingentSearch />        <= no prop values injected here!
```

/components/categoryComponents/CategoriesIntelligentSearch.jsx: though has the answer to the mystery. How does the `CategoriesIntellingentSearch` get those above underlined prop values? The hints start already from the imports:

```
// function from a module for connecting those two
import { connect } from 'react-redux';

// action dispatcher, the function we created before!
import {categoriesSearchByKeyword} from '../../actions/category';

// a function returning a definition object describing how to connect the Redux store
// state values to this components props!
const mapStateToProps = state => ({
  categories: state.categories,
});

// a function returning a definition object describing how to connect the local function
// calls to Redux code function calls (async).
const mapDispatchToProps = dispatch => ({
  categoriesSearchByKeywordLocal: (keyword) => {
    dispatch(categoriesSearchByKeyword(keyword));
  },
});

// Finally, what will be returned is not the React component, but the React component
// after it has bee connected to the Redux store data, and our Redux code functions!
export default connect(mapStateToProps, mapDispatchToProps)(CategoriesIntellingentSearch);
```

All parts touched, but most likely needs improvements for the explanations.