Universal React Redux Boilerplate

Loading up the state on Server

- take a component, **Blog** –> `src/modules/Blog/Blog.js`
- **onEnter** we want to fetch data from an API : `Blog.onEnter = ({dispatch}) => dispatch(fetchArticles())`
- `fetchArticles()` will make a request to a server, and on successful response, return the JSON
- Back on our component, **`Blog.js`**, we need to define what KEY in the STATE will be updated with that info… look in the ROOT reducer, `src/reducer/js` to see the ROOT reducers. They exist inside the `combineReducers()` function.
    - so in the `Blog.js` component folder… open up its `redux.js` file. You will again see `combineReducers()` and that is all the data that will exist within the root `blog` key in your Redux store.

```
const reducer = combineReducers({
  articles,
  article,
  listOfThings: theList
})
```

- each of these `keys` has a reducer attached to it, in which it create the initial state, and–based on the ACTION triggered– goes through a `switchcase` statement to check the `type` fired off in the action and then return data… typically `action.response` which is a JSON response from an isomorphic-fetch
    - Now we need to allow our component, `Blog.js` to access the state as props to use in the UI, so far we have something like this (Blog.js file)*

```
Blog.onEnter = ({ dispatch }) => dispatch(fetchArticles())

const mapStateToProps = state => ({
  articles: getArticles(state),
  listOfThings: getDataExample(state)
})

export default connect(mapStateToProps)(Blog)

```

- also notice the value for `listOfThings:` above. We include a function `getDataExample(state)` with the state to try and get a sliver of the state.
- … in the `redux.js` file, we export the const for use in `Blog.js` which looks like this:

```
export const getDataExample = state => state.blog.listOfThings
```

- Now let's look at the `fetchArticles()` function(actually called the Action Creator).. this function just triggers the `fetchAction()` Action but with a specific API endpoint which returns a TYPE and a RESPONSE known as the payload the reducer. Remember, reducers should be pure functions, no logic should go in them!

```
export const fetchArticles = () => fetchAction(
  '/api/articles',
  [FETCH_ARTICLES_REQUEST, FETCH_ARTICLES_SUCCESS, FETCH_ARTICLES_FAILURE]
)
```

it makes a request to a URL and returns the possible response types.
- always dispatch `REQUEST` type
- get the json `response` and then trigger the action.types.
- our `fetchAction()` action will rename the types to just `REQUETST`, `SUCCESS`, and `FAILURE`
- prefixing them is necessary to make sure that all reducers have a different `type` since if we remember: all reducers always get fired off and we use a switchcase statement to check if the `type` was triggered.
- Now we are looking in **`src/modules/Blog/redux.js`**

```
const fetchAction = (url, types) => dispatch => {
  const [REQUEST, SUCCESS, FAILURE] = types

  if (__SERVER__) {
    url = `http://localhost:3000${ url }`
  }

  dispatch({
    type: REQUEST
  })

  return fetch(url)
    .then(
      response => response.json()
    )
    .then(
      response => dispatch({
        type: SUCCESS,
        response
      }),
      error => dispatch({
        type: FAILURE,
        message: error.message
      })
    )
}
```

- in the above let's say we trigger fetchArticles… `BlogArticle.onEnter = ({ dispatch }, { params }) => dispatch(fetchArticle(params.slug))`. It will run the above and on `SUCCESS` will `switchcase` until the type `FETCH_ARTICLES_SUCCSES` is found and then update the state with the data from `FETCH_ARTICLES_SUCCSES`
- we also need to map the state to props so the component can use it

```
const mapStateToProps = state => ({
  articles: getArticles(state),
  listOfThings: getDataExample(state)
})
```