**The given Candidates AJAX example explained**

The given example already demonstrates two things:

1) How to “orchestrate”/combine/form one view (on single-page app, SPA) based on **many independent React components**

2) How those components are independent, but share the same model, via AJAX to web service back-end. That example uses e.g. timer triggered refreshing of one component’s (candidate list) state from the back-end.

That example works, but **you’d need to set** the back-end URL and port number to match yours, plus run also that back-end Node.js server e.g. in same machine, localhost, but on different port from the create-react-app front end development environment.

(Yes, create-react-app project and tool compilation is a development environment, the final front-end React app will not have Node.js and npm. But the development environment produces html and JS that can be deployed e.g. to a static front-end server to be the front-end.)

Create the first create-react-app **candidates-ajax** with which you just test the given example without changing more than port number and URL.

Create a create-react-app project (call it e.g. **candidates-redux-solo**) and then download the given folder structure and replace all files while pasting the downloaded changed and added files to the pristine create-react-app project.

**1. What to change in the given project to prepare for Redux?**

Even if this example already works and it’s already divided into independent React components, it has some things that should be removed and some that need to be added so that we can have the first own learning project on Redux:

**Remove by changing or commenting out**.

* Let’s remove all AJAX connections to back-end and thus it’s “database” in this version. Instead we will use a hard-coded initial Redux store state, an array of e.g. 3 candidates as the original model.

**Redux complication**

Like we often do in software development, we split the code into many parts that work together. E.g. into many functions that might call each other. Or objects that are related to each other. We create code that looks to be too complicated for the task, but at the end we realized how modularity and Single Responsibility Principle (SRP) helped us create code that is modifiable and orchestrateable and doesn’t have annoying spaghetti dependencies.

The same applies to Redux. The code consists of seemingly too many steps, but **the aim of Redux** is to be able to use the same Action types, Action creators and the same storage and thus common state shared by as many components as is needed, with common messages sent to it!

So hang in there! Furthermore you will possibly create unforeseen abstract structures to your mind.

**Redux programming steps**

1. Deciding upon the **Application state** that we want to keep in front-end side, in the **Redux Store**. Remember that Redux is not the business data model store, which will be in the back-end database. In this case the front-end components will need a JavaScript array (***candidates***) of zero to many single ***candidate*** objects.
2. Install react-dom, redux, and react-redux modules with npm
3. Setup the Redux store ( imports, setup the const store in Index.js, wrap the App module inside Provider)
4. React components we already have, they just might have to be modified a bit
5. Defining the **Action types** in src/actions/actionTypes.js Action types are just text literals listing the limited standard set of actions that the front-end allows the components to do. E.g. ADD\_CANDIDATE (a new candidate added to the front-end, and in the case of AJAX also immediately offered to the back-end to be accepted), FETCH\_CANDIDATE\_LIST, FIND\_CANDIDATE\_BY\_NAME, DELETE\_CANDIDATE, etc. whatever the UI is able to do.
6. Defining the **Action Creators** (and thus defining also 1. what info is needed in each action how each created **Action** looks like) in src/actions/actions.js
7. Implementing the **Reducers**. (Reducer takes the old state and an Action and knows how to create the new state based on those two) src/reducers/
8. Edit the own React components by defining the state if it needs changes, then add in same ES module, but not inside the component the following:
9. mapStateToProps
10. mapDispatchToProps
11. export default connect (mapStateToProps, mapDispatchToProps) (MyComponent); // This will return a Redux connected component that you can place on any view! (=orchestration)

**HELP**

The Moodle linked, e.g.

* intro material redux-haaga-helia.surge.sh
* Rasmus’ example linked from GitHub

Haaga-Helia library – <https://haaga-helia.finna.fi/> Safari ebooks, and e.g.

Learning Redux

By: Daniel Bugl

Publisher: Packt Publishing

Pub. Date: August 31, 2017

Web ISBN-13: 978-1-78646-953-3

Print ISBN-13: 978-1-78646-239-8

Remember that you need to be file and folder aware! So imports might be written different ways in the given code. Browse your project until familiar with the structure. Check every import you make.