React – Flux

In this lab, you will modify the Kraken app to use Flux and the Flux Dispatcher.

# Objectives

In this lab, you will

* Modify an existing Kraken app
* Add a Flux Dispatcher
* Add a TodoStore which uses the todo-controller
* Add TodoStore CHANGE listenders
* Run the app

# Run the Kraken app to verify correctness

1. Go to the lab/react-app folder and start the application. It should display an app with several components. The <Status> component displays the number of TODO items while the <Todos> component displays the actual TODO application. Start the app and view the page at <http://localhost:8000>

npm install

npm start

# Create the Dispatcher

1. Create the /public/js/dispatcher.js file and insert the following contents:

**var *Dispatcher*** = ***require***(**'flux'**).Dispatcher;  
  
**module**.exports = **new *Dispatcher***();

1. Notice we are using the Facebook Flux Dispatcher.
2. Create the /public/js/constants.js file and insert the following contents:

**module**.exports = {  
 **TODO\_CREATE**: 1,  
 **TODO\_FIND\_ALL**: 2,  
 **TODO\_UPDATE**: 3,  
 **TODO\_DELETE**: 4  
};

1. These are the actionType values we use in the TodoStore.
2. Create the /public/js/todo-store.js file and insert the following contents:

**var *Dispatcher*** = ***require***(**'./dispatcher.js'**);  
**var *constants*** = ***require***(**'./constants.js'**);  
**var *controller*** = ***require***(**'./todo-controller.js'**);  
**var *EventEmitter*** = ***require***(**'events'**).EventEmitter;  
**var *assign*** = ***require***(**'object-assign'**);  
  
**var *CHANGE\_EVENT*** = **'change'**;  
  
**var *todos*** = [];

1. The above are the requirements for the TodoStore. The Dispatcher and Constants we just looked at. The Controller has not changed. The EventEmitter allows the clients of the TodoStore to listen for events. The Assign allows the TodoStore to add the EventEmitter’s methods (like extending in an OO Language).
2. Append the following to the end of the same file.

**function** *emitChangeEvent*() {  
 ***TodoStore***.emitChange();  
}  
  
**var *TodoStore*** = ***assign***({}, ***EventEmitter***.**prototype**, {  
  
 getAll: **function** () {  
 **return *todos***;  
 },  
  
 emitChange: **function**() {  
 **this**.emit( ***CHANGE\_EVENT*** );  
 },  
  
 registerChangeListener: **function** (callback) {  
 **this**.on(***CHANGE\_EVENT***, callback);  
 },  
  
 removeChangeListener: **function** (callback) {  
 **this**.removeListener(***CHANGE\_EVENT***, callback);  
 }  
  
});

1. The above is the store. It only allows the clients to get all of the TODO list and register event listeners. When the TODO list changes (via the Dispatcher shown later), the store will call all the registered event listeners. The event listener will usually call getAll() to grab the current list of TODO items.
2. Append the following to the same file.

***Dispatcher***.register(**function** (action) {  
  
 **switch** (action.**actionType**) {  
  
 **case *constants***.**TODO\_CREATE**:  
 ***controller***.create(action.**content**, **function** (results) {  
 ***todos*** = results;  
 *emitChangeEvent*();  
 });  
 **break**;  
  
 **case *constants***.**TODO\_DELETE**:  
 ***controller***.delete(action.**id**, **function** (results) {  
 ***todos*** = results;  
 *emitChangeEvent*();  
 });  
 **break**;  
  
 **case *constants***.**TODO\_FIND\_ALL**:  
 ***controller***.findAll(**function** (results) {  
 ***todos*** = results;  
 *emitChangeEvent*();  
 });  
 **break**;  
  
 **case *constants***.**TODO\_UPDATE**:  
 **var** item = {  
 **id**: action.**id**,  
 **content**: action.**content** };  
 ***controller***.update(item, **function** (results) {  
 ***todos*** = results;  
 *emitChangeEvent*();  
 });  
 **break**;  
  
 }  
});  
  
**module**.exports = ***TodoStore***;

1. The above registers the TodoStore with the Dispatcher. The client invokes the Dispatcher with the appropriate action object. The Dispatcher uses the switch statement to invoke the appropriate controller method (which in turn uses AJAX to call the server). It saves the results into the global variable, todos, and instructs the TodoStore to fire the EventEmitter. NOTE: each action object has a different set of properties based on the requirements of the AJAX calls. Some require just the id, some require the entire TODO item, etc.

# Integrate Flux with the Components

Of the four components, two of them (home.jsx and layout.jsx) do not change. The <Status> component (/public/views/status.jsx) now registers an event listener so that it knows when the TODO list changes, but does NOT directly change the list. The <Todos> component (/public/views/todos.jsx) now registers an event listener AND refactors all controller calls to go through the Dispatcher.

1. Change the <TodoList> component of the /public/views/todos.jsx to the following:

**var *TodoList*** = **module**.exports = ***React***.**createClass**({  
 getInitialState() {  
 **return** {  
 **todos**: []  
 }  
 },  
  
 componentDidMount() {  
 ***TodoStore***.registerChangeListener(**this**.\_change);  
 ***Dispatcher***.dispatch({**actionType**: ***Constants***.**TODO\_FIND\_ALL**});  
 },  
  
 \_change() {  
 **this**.*setState*( { **todos**: ***TodoStore***.getAll() });  
 },  
  
 render: **function** *render*() {  
 **var** self = **this**;  
  
 **return** (  
 <**div id="layout"**>  
 <**h1 id="page-title"**>Express Todo</**h1**>  
 <**div id="list"**>  
  
 <**TodoInput addTodo=**{**this**.addTodo} />  
  
 {**this**.state.**todos**.**map**( **function**(todo, idx) {  
 **return** ( <**TodoItem key=**{idx} **todo=**{todo} **delete=**{self.delete} /> );  
 })}  
  
 </**div**>  
 </**div**>  
 )  
 },  
 addTodo( content ) {  
 ***Dispatcher***.dispatch({**actionType**: ***Constants***.**TODO\_CREATE**, **content**: content});},  
 **delete**(id) {  
 ***Dispatcher***.dispatch({**actionType**: ***Constants***.**TODO\_DELETE**, **id**: id});  
 }  
});

1. Notice the calls to register the change listener (in green).
2. Notice the calls to the Dispatcher (in yellow). The action parameter has different properties based on the actionType.
3. Change the require() statements in the file to:

**var *React*** = ***require***(**'react'**);  
  
**var *Constants*** = ***require***(**'../js/constants'**);  
**var *TodoStore*** = ***require***(**'../js/todo-store'**);  
**var *Dispatcher*** = ***require***(**'../js/dispatcher.js'**);

1. This loads the interface to the Dispatcher.
2. Add the following two methods to the contents to the /public/views/status.jsx file to the following:

componentDidMount() {  
 ***TodoStore***.registerChangeListener(**this**.\_change);  
 **this**.\_change();  
},  
  
\_change() {  
 **this**.setState( { **number**: ***TodoStore***.getAll().**length** } );  
},

1. This registers the change listener and updates the this.state.number value each time the TODO list changes.
2. Install the new modules we use with:

npm install events flux object-assign --save

1. Start the server and open the browser to <http://localhost:8000> to see the page.
2. Now, each time you add or remove a TODO item, the status component changes its number and the TODO app works normally.

Congratulations. You have completed this lab.