# React + Redux - User Registration and Login Tutorial & Example

Sep 27 2017 - Updated to **React 16**.

In this tutorial we'll cover how to implement user registration and login functionality with React and Redux. The tutorial demo is a React + Redux Boilerplate application that uses JWT authentication, it's based on the code from a real world secure web application I developed for a law firm in Sydney recently.

The project is available on GitHub at <https://github.com/cornflourblue/react-redux-registration-login-example>.

The boilerplate code uses a fake / mock backend that uses browser local storage for managing application data, to switch to a real backend api you just have to remove a couple of lines of code from the main react entry file (/src/index.jsx).

## React + Redux Auth Boilerplate Demo

Here's the login tutorial demo application in action on plunker.

(See on Plunker at <http://plnkr.co/edit/eXV3Wa?p=preview>)

Update History:

* 23 Oct 2017 - For an extended example with a real back-end API built with ASP.NET Core go to [React + Redux with ASP.NET Core 2.0 - Login & Registration Tutorial & Example](https://www.pointblankdevelopment.com.au/blog/124/react-redux-with-aspnet-core-20-login-registration-tutorial-example)
* 27 Sep 2017 - Updated to **React 16.0.0**
* 16 Sep 2017 - For the same example built with Angular 2/4 check out [Angular 2/4 User Registration and Login Example & Tutorial](http://jasonwatmore.com/post/2016/09/29/angular-2-user-registration-and-login-example-tutorial)
* 16 Sep 2017 - Built with **React 15.4.2**

## Running the React + Redux Boilerplate Demo Locally

1. Install NodeJS and NPM from <https://nodejs.org/en/download/>.
2. Download or clone the project source code from <https://github.com/cornflourblue/react-redux-registration-login-example>
3. Install all required npm packages by running npm install from the command line in the project root folder (where the package.json is located).
4. Start the application by running npm start from the command line in the project root folder, this will launch a browser displaying the application.

## React + Redux Project Structure

All source code for the React + Redux tutorial app is located in the /src folder. Inside the src folder there is a folder per feature (App, HomePage, LoginPage, RegisterPage) and a bunch of folders for non-feature code that can be shared across different parts of the app (\_actions, \_components, \_constants, \_helpers, \_reducers, \_services).

I prefixed non-feature folders with an underscore "\_" to group them together and make it easy to distinguish between features and non-features, it also keeps the project folder structure shallow so it's quick to see everything at a glance from the top level and to navigate around the project.

The index.js files in each folder are barrel files that group all the exported modules together so they can be imported using the folder path instead of the full module path and to enable importing multiple modules in a single import (e.g. import { userActions, alertActions } from '../\_actions').

Click any of the below links to jump down to a description of each file along with it's code:

* src
  + [\_actions](http://jasonwatmore.com/#actions-folder)
    - [alert.actions.js](http://jasonwatmore.com/#alert-actions-js)
    - index.js
    - [user.actions.js](http://jasonwatmore.com/#user-actions-js)
  + [\_components](http://jasonwatmore.com/#components-folder)
    - index.js
    - [PrivateRoute.jsx](http://jasonwatmore.com/#private-route-jsx)
  + [\_constants](http://jasonwatmore.com/#constants-folder)
    - [alert.constants.js](http://jasonwatmore.com/#alert-constants-js)
    - index.js
    - [user.constants.js](http://jasonwatmore.com/#user-constants-js)
  + [\_helpers](http://jasonwatmore.com/#helpers-folder)
    - [auth-header.js](http://jasonwatmore.com/#auth-header-js)
    - [fake-backend.js](http://jasonwatmore.com/#fake-backend-js)
    - [history.js](http://jasonwatmore.com/#history-js)
    - index.js
    - [store.js](http://jasonwatmore.com/#store-js)
  + [\_reducers](http://jasonwatmore.com/#reducers-folder)
    - [alert.reducer.ts](http://jasonwatmore.com/#alert-reducer-js)
    - [authentication.reducer.js](http://jasonwatmore.com/#authentication-reducer-js)
    - index.js
    - [registration.reducer.js](http://jasonwatmore.com/#registration-reducer-js)
    - [users.reducer.js](http://jasonwatmore.com/#users-reducer-js)
  + [\_services](http://jasonwatmore.com/#services-folder)
    - index.js
    - [user.service.js](http://jasonwatmore.com/#user-service-js)
  + [App](http://jasonwatmore.com/#app-folder)
    - [App.jsx](http://jasonwatmore.com/#app-jsx)
    - index.js
  + [HomePage](http://jasonwatmore.com/#home-page-folder)
    - [HomePage.jsx](http://jasonwatmore.com/#home-page-jsx)
    - index.js
  + [LoginPage](http://jasonwatmore.com/#login-page-folder)
    - index.js
    - [LoginPage.jsx](http://jasonwatmore.com/#login-page-jsx)
  + [RegisterPage](http://jasonwatmore.com/#register-page-folder)
    - index.js
    - [RegisterPage.jsx](http://jasonwatmore.com/#register-page-jsx)
  + [index.html](http://jasonwatmore.com/#index-html)
  + [index.jsx](http://jasonwatmore.com/#index-jsx)
* package.json
* webpack.config.js

## Redux Actions Folder

**Path: /src/\_actions**

The \_actions folder contains all the Redux action creators for the project, I've organised the action creators into different files by action type (e.g. user actions, alert actions etc).

If you're not familiar with Redux actions or action creators you can learn about them at <http://redux.js.org/docs/basics/Actions.html>.

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Alert Action Creators

**Path: /src/\_actions/alert.actions.js**

Contains Redux action creators for actions related to alerts / toaster notifications in the application. For example to display a success alert message with the text 'Registration Successful' you can call dispatch(alertActions.success('Registration successful'));.

I've wrapped the action methods in an alertActions object at the top of the file so it's easy to see all available actions at a glance and simplifies importing them into other files. The implementation details for each action creator are placed in the below functions.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import { alertConstants } from '../\_constants';    export const alertActions = {      success,      error,      clear  };    function success(message) {      return { type: alertConstants.SUCCESS, message };  }    function error(message) {      return { type: alertConstants.ERROR, message };  }    function clear() {      return { type: alertConstants.CLEAR };  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux User Action Creators

**Path: /src/\_actions/user.actions.js**

Contains Redux action creators for actions related to users. Public action creators are exposed via the userActions object at the top of the file and function implementations are located below, I like this structure because you can quickly see all of the actions that are available.

Most of the actions for users are async actions that are made up of multiple sub actions, this is because they have to make an http request and wait for the response before completing. Async actions typically dispatch a request action before performing an async task (e.g. an http request), and then dispatch a success or error action based on the result of the async task.

For example the login() action creator performs the following steps:

1. dispatches a LOGIN\_REQUEST action with dispatch(request({ username }));
2. calls the async task userService.login(username, password)
3. dispatches a LOGIN\_SUCCESS with dispatch(success(user)); if login was successful, or dispatches a LOGIN\_FAILURE action with dispatch(failure(error)); if login failed

To keep the code tidy I've put sub action creators into nested functions within each async action creator function. For example the login()function contains 3 nested action creator functions for request(), success() and failure() that return the actions LOGIN\_REQUEST, LOGIN\_SUCCESS and LOGIN\_FAILURE respectively. Putting the sub action creators into nested functions also allows me to give them standard names like request, success and failure without clashing with other function names because they only exist within the scope of the parent function.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102 | import { userConstants } from '../\_constants';  import { userService } from '../\_services';  import { alertActions } from './';  import { history } from '../\_helpers';    export const userActions = {      login,      logout,      register,      getAll,      delete: \_delete  };    function login(username, password) {      return dispatch => {          dispatch(request({ username }));            userService.login(username, password)              .then(                  user => {                      dispatch(success(user));                      history.push('/');                  },                  error => {                      dispatch(failure(error));                      dispatch(alertActions.error(error));                  }              );      };        function request(user) { return { type: userConstants.LOGIN\_REQUEST, user } }      function success(user) { return { type: userConstants.LOGIN\_SUCCESS, user } }      function failure(error) { return { type: userConstants.LOGIN\_FAILURE, error } }  }    function logout() {      userService.logout();      return { type: userConstants.LOGOUT };  }    function register(user) {      return dispatch => {          dispatch(request(user));            userService.register(user)              .then(                  user => {                      dispatch(success());                      history.push('/login');                      dispatch(alertActions.success('Registration successful'));                  },                  error => {                      dispatch(failure(error));                      dispatch(alertActions.error(error));                  }              );      };        function request(user) { return { type: userConstants.REGISTER\_REQUEST, user } }      function success(user) { return { type: userConstants.REGISTER\_SUCCESS, user } }      function failure(error) { return { type: userConstants.REGISTER\_FAILURE, error } }  }    function getAll() {      return dispatch => {          dispatch(request());            userService.getAll()              .then(                  users => dispatch(success(users)),                  error => {                      dispatch(failure(error));                      dispatch(alertActions.error(error))                  }              );      };        function request() { return { type: userConstants.GETALL\_REQUEST } }      function success(users) { return { type: userConstants.GETALL\_SUCCESS, users } }      function failure(error) { return { type: userConstants.GETALL\_FAILURE, error } }  }    // prefixed function name with underscore because delete is a reserved word in javascript  function \_delete(id) {      return dispatch => {          dispatch(request(id));            userService.delete(id)              .then(                  user => {                      dispatch(success(id));                  },                  error => {                      dispatch(failure(id, error));                  }              );      };        function request(id) { return { type: userConstants.DELETE\_REQUEST, id } }      function success(id) { return { type: userConstants.DELETE\_SUCCESS, id } }      function failure(id, error) { return { type: userConstants.DELETE\_FAILURE, id, error } }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React Components Folder

**Path: /src/\_components**

The \_components folder contains shared React components that can be used anywhere in the application.

[Back to top](http://jasonwatmore.com/#projectstructure)

## React Private Route Component

**Path: /src/\_components/PrivateRoute.jsx**

The react private route component renders a route component if the user is logged in, otherwise it redirects the user to the /login page.

The way it checks if the user is logged in is by checking that there is a user object in local storage. While it's possible to bypass this check by manually adding an object to local storage using browser dev tools, this would only give access to the client side component, it wouldn't give access to any real secure data from the server api because a valid authentication token (JWT) is required for this.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import React from 'react';  import { Route, Redirect } from 'react-router-dom';    export const PrivateRoute = ({ component: Component, ...rest }) => (      <Route {...rest} render={props => (          localStorage.getItem('user')              ? <Component {...props} />              : <Redirect to={{ pathname: '/login', state: { from: props.location } }} />      )} />  ) |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Action Constants Folder

**Path: /src/\_constants**

The \_constants folder contains all of the redux action type constants used by redux action creators and reducers. It could be used for any other constants in the project as well, it doesn't have to be only for redux action types.

I decided to put redux action constants into their own files (rather than the same files as redux actions) to simplify my redux action files and keep them focused on one thing.

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Alert Action Constants

**Path: /src/\_constants/alert.constants.js**

The alert constants object contains the redux alert action types used to display and clear alerts in the react application.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | export const alertConstants = {      SUCCESS: 'ALERT\_SUCCESS',      ERROR: 'ALERT\_ERROR',      CLEAR: 'ALERT\_CLEAR'  }; |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux User Action Constants

**Path: /src/\_constants/user.constants.js**

The user constants object contains the redux user action types that can be dispatched in the react application, async actions that perform http requests involve a request followed by a success or error response, so each of these three steps is represented by a redux action.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | export const userConstants = {      REGISTER\_REQUEST: 'USERS\_REGISTER\_REQUEST',      REGISTER\_SUCCESS: 'USERS\_REGISTER\_SUCCESS',      REGISTER\_FAILURE: 'USERS\_REGISTER\_FAILURE',        LOGIN\_REQUEST: 'USERS\_LOGIN\_REQUEST',      LOGIN\_SUCCESS: 'USERS\_LOGIN\_SUCCESS',      LOGIN\_FAILURE: 'USERS\_LOGIN\_FAILURE',        LOGOUT: 'USERS\_LOGOUT',        GETALL\_REQUEST: 'USERS\_GETALL\_REQUEST',      GETALL\_SUCCESS: 'USERS\_GETALL\_SUCCESS',      GETALL\_FAILURE: 'USERS\_GETALL\_FAILURE',        DELETE\_REQUEST: 'USERS\_DELETE\_REQUEST',      DELETE\_SUCCESS: 'USERS\_DELETE\_SUCCESS',      DELETE\_FAILURE: 'USERS\_DELETE\_FAILURE'  }; |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux Helpers Folder

**Path: /src/\_helpers**

The helpers folder contains all the bits and pieces that don't fit into other folders but don't justify having a folder of their own.

[Back to top](http://jasonwatmore.com/#projectstructure)

## React Auth Header

**Path: /src/\_helpers/auth-header.js**

Auth header is a helper function that returns an HTTP Authorization header containing the Json Web Token (JWT) of the currently logged in user from local storage. If the user isn't logged in an empty object is returned.

The auth header is used to make authenticated HTTP requests to the server api using JWT authentication.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | export function authHeader() {      // return authorization header with jwt token      let user = JSON.parse(localStorage.getItem('user'));        if (user && user.token) {          return { 'Authorization': 'Bearer ' + user.token };      } else {          return {};      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React Fake / Mock Backend

**Path: /src/\_helpers/fake-backend.js**

The fake backend is used for running the example api without a server api (backend-less). It monkey patches the fetch() function to intercept certain api requests and mimic the behaviour of a real api by managing data in browser local storage. Any requests that aren't intercepted get passed through to the real fetch() function.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129 | // array in local storage for registered users  let users = JSON.parse(localStorage.getItem('users')) || [];    export function configureFakeBackend() {      let realFetch = window.fetch;      window.fetch = function (url, opts) {          return new Promise((resolve, reject) => {              // wrap in timeout to simulate server api call              setTimeout(() => {                    // authenticate                  if (url.endsWith('/users/authenticate') && opts.method === 'POST') {                      // get parameters from post request                      let params = JSON.parse(opts.body);                        // find if any user matches login credentials                      let filteredUsers = users.filter(user => {                          return user.username === params.username && user.password === params.password;                      });                        if (filteredUsers.length) {                          // if login details are valid return user details and fake jwt token                          let user = filteredUsers[0];                          let responseJson = {                              id: user.id,                              username: user.username,                              firstName: user.firstName,                              lastName: user.lastName,                              token: 'fake-jwt-token'                          };                          resolve({ ok: true, json: () => responseJson });                      } else {                          // else return error                          reject('Username or password is incorrect');                      }                        return;                  }                    // get users                  if (url.endsWith('/users') && opts.method === 'GET') {                      // check for fake auth token in header and return users if valid, this security is implemented server side in a real application                      if (opts.headers && opts.headers.Authorization === 'Bearer fake-jwt-token') {                          resolve({ ok: true, json: () => users });                      } else {                          // return 401 not authorised if token is null or invalid                          reject('Unauthorised');                      }                        return;                  }                    // get user by id                  if (url.match(/\/users\/\d+$/) && opts.method === 'GET') {                      // check for fake auth token in header and return user if valid, this security is implemented server side in a real application                      if (opts.headers && opts.headers.Authorization === 'Bearer fake-jwt-token') {                          // find user by id in users array                          let urlParts = url.split('/');                          let id = parseInt(urlParts[urlParts.length - 1]);                          let matchedUsers = users.filter(user => { return user.id === id; });                          let user = matchedUsers.length ? matchedUsers[0] : null;                            // respond 200 OK with user                          resolve({ ok: true, json: () => user});                      } else {                          // return 401 not authorised if token is null or invalid                          reject('Unauthorised');                      }                        return;                  }                    // register user                  if (url.endsWith('/users/register') && opts.method === 'POST') {                      // get new user object from post body                      let newUser = JSON.parse(opts.body);                        // validation                      let duplicateUser = users.filter(user => { return user.username === newUser.username; }).length;                      if (duplicateUser) {                          reject('Username "' + newUser.username + '" is already taken');                          return;                      }                        // save new user                      newUser.id = Math.max(...users.map(user => user.id)) + 1;                      users.push(newUser);                      localStorage.setItem('users', JSON.stringify(users));                        // respond 200 OK                      resolve({ ok: true, json: () => ({}) });                        return;                  }                    // delete user                  if (url.match(/\/users\/\d+$/) && opts.method === 'DELETE') {                      // check for fake auth token in header and return user if valid, this security is implemented server side in a real application                      if (opts.headers && opts.headers.Authorization === 'Bearer fake-jwt-token') {                          // find user by id in users array                          let urlParts = url.split('/');                          let id = parseInt(urlParts[urlParts.length - 1]);                          for (let i = 0; i < users.length; i++) {                              let user = users[i];                              if (user.id === id) {                                  // delete user                                  users.splice(i, 1);                                  localStorage.setItem('users', JSON.stringify(users));                                  break;                              }                          }                            // respond 200 OK                          resolve({ ok: true, json: () => ({}) });                      } else {                          // return 401 not authorised if token is null or invalid                          reject('Unauthorised');                      }                        return;                  }                    // pass through any requests not handled above                  realFetch(url, opts).then(response => resolve(response));                }, 500);          });      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React Router History

**Path: /src/\_helpers/history.js**

The history is a custom history object used by the React Router, the reason I used a custom history object instead of the built into React Router is to enable redirecting users from outside React components, for example from the user actions after successful login or registration.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3 | import { createBrowserHistory } from 'history';    export const history = createBrowserHistory(); |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Store

**Path: /src/\_helpers/store.js**

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import { createStore, applyMiddleware } from 'redux';  import thunkMiddleware from 'redux-thunk';  import { createLogger } from 'redux-logger';  import rootReducer from '../\_reducers';    const loggerMiddleware = createLogger();    export const store = createStore(      rootReducer,      applyMiddleware(          thunkMiddleware,          loggerMiddleware      )  ); |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Reducers Folder

**Path: /src/\_reducers**

The \_reducers folder contains all the Redux reducers for the project, each reducer updates a different part of the application state in response to dispatched redux actions.

If you're not familiar with Redux reducers you can learn about them at <http://redux.js.org/docs/basics/Reducers.html>.

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Alert Reducer

**Path: /src/\_reducers/alert.reducer.js**

The redux alert reducer manages the application state for alerts / toaster notifications, it updates state when an alert action is dispatched from anywhere in the application, for example when an alertConstants.SUCCESS action is dispatched, the reducer updates the alert state to an object with type: 'alert-success' and message: action.message.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import { alertConstants } from '../\_constants';    export function alert(state = {}, action) {    switch (action.type) {      case alertConstants.SUCCESS:        return {          type: 'alert-success',          message: action.message        };      case alertConstants.ERROR:        return {          type: 'alert-danger',          message: action.message        };      case alertConstants.CLEAR:        return {};      default:        return state    }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Authentication Reducer

**Path: /src/\_reducers/authentication.reducer.js**

The redux authentication reducer manages the state related to login (and logout) actions, on successful login the current user object and a loggedIn flag are stored in the authentication section of the application state. On logout or login failure the authentication state is set to an empty object, and during login (between login request and success/failure) the authentication state has a loggingIn flag set to true and a user object with the details of the user that is attempting to login.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import { userConstants } from '../\_constants';    let user = JSON.parse(localStorage.getItem('user'));  const initialState = user ? { loggedIn: true, user } : {};    export function authentication(state = initialState, action) {    switch (action.type) {      case userConstants.LOGIN\_REQUEST:        return {          loggingIn: true,          user: action.user        };      case userConstants.LOGIN\_SUCCESS:        return {          loggedIn: true,          user: action.user        };      case userConstants.LOGIN\_FAILURE:        return {};      case userConstants.LOGOUT:        return {};      default:        return state    }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Registration Reducer

**Path: /src/\_reducers/registration.reducer.js**

The redux registration reducer manages the registration section of the application state, as you can see there isn't much to it, on registration request it just sets a registering flag set to true which the RegisterPage uses to show the loading spinner. On register success or failure it clears the registration state.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import { userConstants } from '../\_constants';    export function registration(state = {}, action) {    switch (action.type) {      case userConstants.REGISTER\_REQUEST:        return { registering: true };      case userConstants.REGISTER\_SUCCESS:        return {};      case userConstants.REGISTER\_FAILURE:        return {};      default:        return state    }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Redux Users Reducer

**Path: /src/\_reducers/users.reducer.js**

The redux users reducer manages the users section of the application state which is used by the HomePage to display a list of users and enable deleting of users.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | import { userConstants } from '../\_constants';    export function users(state = {}, action) {    switch (action.type) {      case userConstants.GETALL\_REQUEST:        return {          loading: true        };      case userConstants.GETALL\_SUCCESS:        return {          items: action.users        };      case userConstants.GETALL\_FAILURE:        return {          error: action.error        };      case userConstants.DELETE\_REQUEST:        // add 'deleting:true' property to user being deleted        return {          ...state,          items: state.items.map(user =>            user.id === action.id              ? { ...user, deleting: true }              : user          )        };      case userConstants.DELETE\_SUCCESS:        // remove deleted user from state        return {          items: state.items.filter(user => user.id !== action.id)        };      case userConstants.DELETE\_FAILURE:        // remove 'deleting:true' property and add 'deleteError:[error]' property to user        return {          ...state,          items: state.items.map(user => {            if (user.id === action.id) {              // make copy of user without 'deleting:true' property              const { deleting, ...userCopy } = user;              // return copy of user with 'deleteError:[error]' property              return { ...userCopy, deleteError: action.error };            }              return user;          })        };      default:        return state    }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux Services Folder

**Path: /src/\_services**

The \_services layer handles all http communication with backend apis for the application, each service encapsulates the api calls for a content type (e.g. users) and exposes methods for performing various operations (e.g. CRUD operations). Services can also have methods that don't wrap http calls, for example the userService.logout() method just removes an item from local storage.

I like wrapping http calls and implementation details in a services layer, it provides a clean separation of concerns and simplifies the redux actions (and other modules) that use the services.

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux User Service

**Path: /src/\_services/user.service.js**

The user service encapsulates all backend api calls for performing CRUD operations on user data, as well as logging and out of the example application. The service methods are exported via the userService object at the top of the file, and the implementation of each method is located in the function declarations below.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98 | import { authHeader } from '../\_helpers';    export const userService = {      login,      logout,      register,      getAll,      getById,      update,      delete: \_delete  };    function login(username, password) {      const requestOptions = {          method: 'POST',          headers: { 'Content-Type': 'application/json' },          body: JSON.stringify({ username, password })      };        return fetch('/users/authenticate', requestOptions)          .then(response => {              if (!response.ok) {                  return Promise.reject(response.statusText);              }                return response.json();          })          .then(user => {              // login successful if there's a jwt token in the response              if (user && user.token) {                  // store user details and jwt token in local storage to keep user logged in between page refreshes                  localStorage.setItem('user', JSON.stringify(user));              }                return user;          });  }    function logout() {      // remove user from local storage to log user out      localStorage.removeItem('user');  }    function getAll() {      const requestOptions = {          method: 'GET',          headers: authHeader()      };        return fetch('/users', requestOptions).then(handleResponse);  }    function getById(id) {      const requestOptions = {          method: 'GET',          headers: authHeader()      };        return fetch('/users/' + \_id, requestOptions).then(handleResponse);  }    function register(user) {      const requestOptions = {          method: 'POST',          headers: { 'Content-Type': 'application/json' },          body: JSON.stringify(user)      };        return fetch('/users/register', requestOptions).then(handleResponse);  }    function update(user) {      const requestOptions = {          method: 'PUT',          headers: { ...authHeader(), 'Content-Type': 'application/json' },          body: JSON.stringify(user)      };        return fetch('/users/' + user.id, requestOptions).then(handleResponse);;  }    // prefixed function name with underscore because delete is a reserved word in javascript  function \_delete(id) {      const requestOptions = {          method: 'DELETE',          headers: authHeader()      };        return fetch('/users/' + id, requestOptions).then(handleResponse);;  }    function handleResponse(response) {      if (!response.ok) {          return Promise.reject(response.statusText);      }        return response.json();  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux App Folder

**Path: /src/App**

The app folder is for react components and other code that is used only by the app component in the tutorial application.

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux App Component

**Path: /src/App/App.jsx**

The app component is the root component for the react tutorial application, it contains the outer html, routes and global alert notification for the example app.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | import React from 'react';  import { Router, Route } from 'react-router-dom';  import { connect } from 'react-redux';    import { history } from '../\_helpers';  import { alertActions } from '../\_actions';  import { PrivateRoute } from '../\_components';  import { HomePage } from '../HomePage';  import { LoginPage } from '../LoginPage';  import { RegisterPage } from '../RegisterPage';    class App extends React.Component {      constructor(props) {          super(props);            const { dispatch } = this.props;          history.listen((location, action) => {              // clear alert on location change              dispatch(alertActions.clear());          });      }        render() {          const { alert } = this.props;          return (              <div className="jumbotron">                  <div className="container">                      <div className="col-sm-8 col-sm-offset-2">                          {alert.message &&                              <div className={`alert ${alert.type}`}>{alert.message}</div>                          }                          <Router history={history}>                              <div>                                  <PrivateRoute exact path="/" component={HomePage} />                                  <Route path="/login" component={LoginPage} />                                  <Route path="/register" component={RegisterPage} />                              </div>                          </Router>                      </div>                  </div>              </div>          );      }  }    function mapStateToProps(state) {      const { alert } = state;      return {          alert      };  }    const connectedApp = connect(mapStateToProps)(App);  export { connectedApp as App }; |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux Home Page Folder

**Path: /src/HomePage**

The home page folder is for react components and other code that is used only by the home page component in the tutorial application.

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux Home Page Component

**Path: /src/HomePage/HomePage.jsx**

The home page component is displayed after signing in to the application, it shows the signed in user's name plus a list of all registered users in the tutorial application. The users are loaded into redux state by dispatching the redux action userActions.getAll() from the componentDidMount() react lifecycle hook.

Users can also be deleted from the user list, when the delete link is clicked it triggers the redux action userActions.delete(id) to be dispatched.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | import React from 'react';  import { Link } from 'react-router-dom';  import { connect } from 'react-redux';    import { userActions } from '../\_actions';    class HomePage extends React.Component {      componentDidMount() {          this.props.dispatch(userActions.getAll());      }        handleDeleteUser(id) {          return (e) => this.props.dispatch(userActions.delete(id));      }        render() {          const { user, users } = this.props;          return (              <div className="col-md-6 col-md-offset-3">                  <h1>Hi {user.firstName}!</h1>                  <p>You're logged in with React!!</p>                  <h3>All registered users:</h3>                  {users.loading && <em>Loading users...</em>}                  {users.items &&                      <ul>                          {users.items.map((user, index) =>                              <li key={user.id}>                                  {user.firstName + ' ' + user.lastName}                                  {                                      user.deleting ? <em> - Deleting...</em>                                      : user.deleteError ? <span className="error"> - ERROR: {user.deleteError}</span>                                      : <span> - <a onClick={this.handleDeleteUser(user.id)}>Delete</a></span>                                  }                              </li>                          )}                      </ul>                  }                  <p>                      <Link to="/login">Logout</Link>                  </p>              </div>          );      }  }    function mapStateToProps(state) {      const { users, authentication } = state;      const { user } = authentication;      return {          user,          users      };  }    const connectedHomePage = connect(mapStateToProps)(HomePage);  export { connectedHomePage as HomePage }; |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux Login Page Folder

**Path: /src/LoginPage**

The login page folder is for react components and other code that is used only by the login page component in the tutorial application.

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux Login Page Component

**Path: /src/LoginPage/LoginPage.jsx**

The login page component renders a login form with username and password fields. It displays validation messages for invalid fields when the user attempts to submit the form. If the form is valid, submitting it causes the userActions.login(username, password) redux action to be dispatched.

In the constructor() function the userActions.logout() redux action is dispatched which logs the user out if they're logged in, this enables the login page to also be used as the logout page.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82 | import React from 'react';  import { Link } from 'react-router-dom';  import { connect } from 'react-redux';    import { userActions } from '../\_actions';    class LoginPage extends React.Component {      constructor(props) {          super(props);            // reset login status          this.props.dispatch(userActions.logout());            this.state = {              username: '',              password: '',              submitted: false          };            this.handleChange = this.handleChange.bind(this);          this.handleSubmit = this.handleSubmit.bind(this);      }        handleChange(e) {          const { name, value } = e.target;          this.setState({ [name]: value });      }        handleSubmit(e) {          e.preventDefault();            this.setState({ submitted: true });          const { username, password } = this.state;          const { dispatch } = this.props;          if (username && password) {              dispatch(userActions.login(username, password));          }      }        render() {          const { loggingIn } = this.props;          const { username, password, submitted } = this.state;          return (              <div className="col-md-6 col-md-offset-3">                  <h2>Login</h2>                  <form name="form" onSubmit={this.handleSubmit}>                      <div className={'form-group' + (submitted && !username ? ' has-error' : '')}>                          <label htmlFor="username">Username</label>                          <input type="text" className="form-control" name="username" value={username} onChange={this.handleChange} />                          {submitted && !username &&                              <div className="help-block">Username is required</div>                          }                      </div>                      <div className={'form-group' + (submitted && !password ? ' has-error' : '')}>                          <label htmlFor="password">Password</label>                          <input type="password" className="form-control" name="password" value={password} onChange={this.handleChange} />                          {submitted && !password &&                              <div className="help-block">Password is required</div>                          }                      </div>                      <div className="form-group">                          <button className="btn btn-primary">Login</button>                          {loggingIn &&                              <img src="data:image/gif;base64," />                          }                          <Link to="/register" className="btn btn-link">Register</Link>                      </div>                  </form>              </div>          );      }  }    function mapStateToProps(state) {      const { loggingIn } = state.authentication;      return {          loggingIn      };  }    const connectedLoginPage = connect(mapStateToProps)(LoginPage);  export { connectedLoginPage as LoginPage }; |

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux Register Page Folder

**Path: /src/RegisterPage**

The register page folder is for react components and other code that is used only by the register page component in the tutorial application.

[Back to top](http://jasonwatmore.com/#projectstructure)

## React + Redux Register Page Component

**Path: /src/RegisterPage/RegisterPage.jsx**

The register page component renders a simple registration form with fields for first name, last name, username and password. It displays validation messages for invalid fields when the user attempts to submit the form. If the form is valid, submitting it causes the userActions.register(user) redux action to be dispatched.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103 | import React from 'react';  import { Link } from 'react-router-dom';  import { connect } from 'react-redux';    import { userActions } from '../\_actions';    class RegisterPage extends React.Component {      constructor(props) {          super(props);            this.state = {              user: {                  firstName: '',                  lastName: '',                  username: '',                  password: ''              },              submitted: false          };            this.handleChange = this.handleChange.bind(this);          this.handleSubmit = this.handleSubmit.bind(this);      }        handleChange(event) {          const { name, value } = event.target;          const { user } = this.state;          this.setState({              user: {                  ...user,                  [name]: value              }          });      }        handleSubmit(event) {          event.preventDefault();            this.setState({ submitted: true });          const { user } = this.state;          const { dispatch } = this.props;          if (user.firstName && user.lastName && user.username && user.password) {              dispatch(userActions.register(user));          }      }        render() {          const { registering  } = this.props;          const { user, submitted } = this.state;          return (              <div className="col-md-6 col-md-offset-3">                  <h2>Register</h2>                  <form name="form" onSubmit={this.handleSubmit}>                      <div className={'form-group' + (submitted && !user.firstName ? ' has-error' : '')}>                          <label htmlFor="firstName">First Name</label>                          <input type="text" className="form-control" name="firstName" value={user.firstName} onChange={this.handleChange} />                          {submitted && !user.firstName &&                              <div className="help-block">First Name is required</div>                          }                      </div>                      <div className={'form-group' + (submitted && !user.lastName ? ' has-error' : '')}>                          <label htmlFor="lastName">Last Name</label>                          <input type="text" className="form-control" name="lastName" value={user.lastName} onChange={this.handleChange} />                          {submitted && !user.lastName &&                              <div className="help-block">Last Name is required</div>                          }                      </div>                      <div className={'form-group' + (submitted && !user.username ? ' has-error' : '')}>                          <label htmlFor="username">Username</label>                          <input type="text" className="form-control" name="username" value={user.username} onChange={this.handleChange} />                          {submitted && !user.username &&                              <div className="help-block">Username is required</div>                          }                      </div>                      <div className={'form-group' + (submitted && !user.password ? ' has-error' : '')}>                          <label htmlFor="password">Password</label>                          <input type="password" className="form-control" name="password" value={user.password} onChange={this.handleChange} />                          {submitted && !user.password &&                              <div className="help-block">Password is required</div>                          }                      </div>                      <div className="form-group">                          <button className="btn btn-primary">Register</button>                          {registering &&                              <img src="data:image/gif;base64," />                          }                          <Link to="/login" className="btn btn-link">Cancel</Link>                      </div>                  </form>              </div>          );      }  }    function mapStateToProps(state) {      const { registering } = state.registration;      return {          registering      };  }    const connectedRegisterPage = connect(mapStateToProps)(RegisterPage);  export { connectedRegisterPage as RegisterPage }; |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Base Index HTML File

**Path: /src/index.html**

The base index html file contains the outer html for the whole tutorial application. When the app is started with npm start, Webpack bundles up all of the react + redux code into a single javascript file and injects it into the body of the page.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>React + Redux - User Registration and Login Example & Tutorial</title>      <link rel="stylesheet" href="<https://netdna.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css>" />      <style>          a { cursor: pointer; }          .help-block { font-size: 12px; }      </style>  </head>  <body>      <div id="app"></div>  </body>  </html> |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Main React Entry File

**Path: /src/index.jsx**

The root index.jsx file bootstraps the react + redux tutorial application by rendering the App component (wrapped in a redux Provider) into the app div element defined in the base index html file above.

The boilerplate application uses a fake / mock backend that stores data in browser local storage, to switch to a real backend api simply remove the fake backend code below the comment // setup fake backend.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import React from 'react';  import { render } from 'react-dom';  import { Provider } from 'react-redux';    import { store } from './\_helpers';  import { App } from './App';    // setup fake backend  import { configureFakeBackend } from './\_helpers';  configureFakeBackend();    render(      <Provider store={store}>          <App />      </Provider>,      document.getElementById('app')  ); |

[Back to top](http://jasonwatmore.com/#projectstructure)

## Web Development Sydney

Feel free to [drop me a line](http://jasonwatmore.com/contact) if you're looking for a React or Redux web developer in Sydney Australia, I also provide remote contracting services for clients outside Sydney.