

Ducks, Re-ducks, Redux-Toolkit

modular approaches for React/Redux app structure

Sergii Zhuravel

he

Si

35



• Industry:

Telecom, IoT, Automotive, CRM

• Location:

Kyiv, UKR

- Position:
 - Team Lead
- Company

Absio

Fan of JavaScript and JS frameworks. I like table tennis, fishing and traveling

Plan to review

•Why good structure for Redux apps is important

•What is wrong with common approaches in Redux apps

•Ducks in details

•Re-ducks and how it differs from Ducks

•What problems Redux Toolkit resolves and how to start to use it.

•Testing of the duckses

Types of state

- Component's (local) State
- Components shared state
- App's (global) state
- UI state
- Cache

Redux in 2021?



×

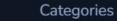
Franciszek Krasnowski • Aug 10 '20

Context API shouldn't be considered as a replacement for Redux. It's an alternative for classic callbacks to parent at most. It's not a state manager. If some dude says that Context will replace Redux, he's living in a fantasy. I can bearly see how to compare those two

♥ 11 likes ♥ Reply

Why React projects still use Redux

...



egories > Utilities > Client & Server Utilities > State Management

10 Best React State Management Libraries

by the Openbase team and community. Learn more



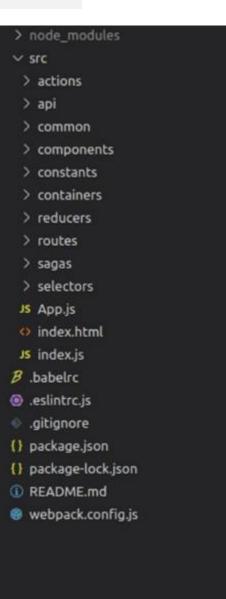


Architecture is important, but today we talk not about it

«bad architecture is the single biggest killer of software project»

Architecture Of Large React Apps: Tools and Techniques

What not to do



Issues:

• Redux artifacts are spread over src folder

 When we need to add new entity in Redux we should open different folders (actions, reducers, sagas, selectors), and add or modify different files

Function vs Feature

🖌 📹 actions

- JS cartActions.js
- JS productActions.js
- JS sessionActions.js
- 🖌 📹 components
 - JS cartItem.js
 - JS footer.js
 - JS header.js
 - JS loginForm.js
 - JS productBox.js
 - JS productButtons.js
 - JS registerForm.js
- 🖌 📹 containers
 - JS home.js
 - JS productDetails.js
 - JS productList.js
 - JS shoppingCart.js
- 🖌 📹 reducers
 - JS cartReducers.js
 - JS productReducers.js
 - JS sessionReducers.js

- 🖌 📹 cart
 - JS cartActions.js
 - JS cartItem.js
 - JS cartReducers.js
 - JS shoppingCart.js
- 🖌 📹 common
 - JS footer.js
 - JS header.js
 - JS home.js
- 🔺 📹 product
 - JS productActions.js
 - JS productBox.js
 - JS productButtons.js
 - JS productDetails.js
 - JS productList.js
 - JS productReducers.js
- 🔺 📹 session
 - JS loginForm.js
 - JS registerForm.js
 - JS sessionActions.js
 - JS sessionReducers.js

Approaches:

- Left side function-first structure of the folders. Function-first means that your top-level directories are named after the purpose of the files inside. So you have: containers, components, actions, red ucers, etc. Problems scaling
- Right side feature-first approach.
 - Feature-first means that the top-level directories are named after the main features of the app: *product*, *cart*, *session*. **Problemsmix of elements of different purpose, it will be harder to change in the future.**

Separate State Management from UI

Think about your application on the long run. Imagine what happens with the codebase when you switch from React to another library. Or think how your codebase would use ReactNative in parallel with the web version.

Scaling your Redux App with ducks

Ducks: Redux Reducer Bundles

- «I find as I am building my redux app, one piece of functionality at a time, I keep needing to add {actionTypes, actions, reducer} tuples for each use case. I have been keeping these in separate files and even separate folders, however 95% of the time, it's only one reducer/actions pair that ever needs their associated actions.
- To me, it makes more sense for these pieces to be bundled together in an isolated module that is self contained, and can even be packaged easily into a library.»

Ducks modular Redux (Erik Rasmussen)

刘 File Edit Selection View Go Run Terminal Help



···· 🛈

🔮 // widgets.js Untitled-1 🔍 G index.html Ducks - example Q // Actions const LOAD = 'my-app/widgets/LOAD'; const CREATE = 'my-app/widgets/CREATE'; const UPDATE = 'my-app/widgets/UPDATE'; const REMOVE = 'my-app/widgets/REMOVE'; P export default function reducer(state = {}, action = {}) { ß switch (action.type) { default: return state; // Action Creators export function loadWidgets() { return { type: LOAD }; export function createWidget(widget) { return { type: CREATE, widget }; export function updateWidget(widget) { return { type: UPDATE, widget }; export function removeWidget(widget) { return { type: REMOVE, widget }; 8 export function getWidget () { return dispatch => get('/widget').then(widget => dispatch(updateWidget(widget))) 53

Ducks- rules

A module...

MUST export default a function called reducer()

MUST export its action creators as functions

MUST have action types in the form npm-module-or-app/reducer/ACTION_TYPE

MAY export its action types as UPPER_SNAKE_CASE, if an external reducer needs to listen for them, or if it is a published reusable library

«Java has jars and beans. Ruby has gems. I suggest we call these reducer bundles "ducks", as in the last syllable of "redux".»

Ducks modular Redux (Erik Rasmussen)

Ducks - use

You can continue to do this:

import { combineReducers } from 'redux';

import * as reducers from './ducks/index';

const rootReducer = combineReducers(reducers);

export default rootReducer;

Ducks-use

You can continue to do this:

import * as widgetActions from './ducks/widgets';

There will be some times when you want to export something other than an action creator. That's okay, too. The rules don't say that you can only export action creators. When that happens, you'll just have to enumerate the action creators that you want.

import {loadWidgets, createWidget, updateWidget, removeWidget} from './ducks/widgets';

// ...

bindActionCreators({loadWidgets, createWidget, updateWidget, removeWidget}, dispatch);

Ducks - examples of implementation

• <u>React Redux Universal Hot Example</u>uses ducks. See /src/redux/modules

• Todomvc using ducks

"The original <u>ducks modular approach</u> is a nice simplification for redux and offers a structured way of adding each new feature in your app.

Yet, we wanted to explore a bit what happens when the app scales. We realized that a single file for a feature becomes too cluttered and hard to maintain on the long run.

This is how *re-ducks* was born. The solution was to split each feature into a duck folder."

Scaling your Redux App with ducks

Inside re-ducks duck folder

duck/	
├── actions.js	
├── index.js	
├── operations.js	
├─ reducers.js	
├── selectors.js	
├─ tests.js	
├─ types.js	
├─ utils.js	

Scaling your Redux App with ducks

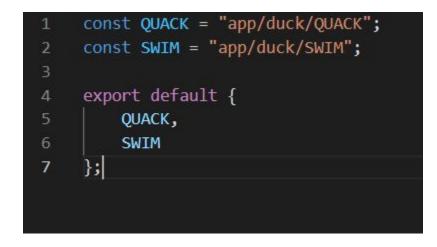
Re-ducks - rules

A duck folder MUST:

- contain the entire logic for handling only ONE concept in your app, ex: product, cart, session, etc.
- have an index.js file that exports according to the original duck rules.
- keep code with similar purpose in the same file, such as reducers, selectors, and actions
- contain the tests related to the duck.

Re-ducks - types

The *types* file contains the names of the actions that you are dispatching in your application. As a good practice, you should try to scope the names based on the feature they belong to. This helps when debugging more complex applications.



Re-ducks -actions

This file contains all the action creator functions.

```
import types from "./types";
     const quack = ( ) => ( {
         type: types.QUACK
     });
     const swim = ( distance ) => ( {
         type: types.SWIM,
         payload: {
             distance
11
12
     });
13
     export default {
14
         swim,
15
         quack
    };
17
```

Re-ducks - operations

To represent chained operations you need a redux *middleware* to enhance the dispatch function. Some popular examples are: <u>redux-thunk</u>, <u>redux-saga</u> or <u>re</u> <u>dux-observable</u>.

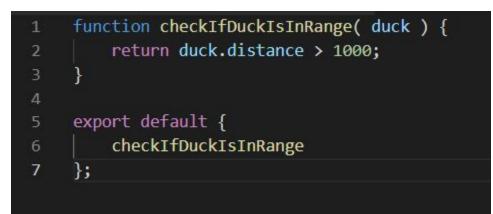
```
import actions from "./actions";
     const simpleQuack = actions.quack;
     // This is a thunk which dispatches multiple actions from actions.js
     const complexQuack = ( distance ) => ( dispatch ) => {
         dispatch( actions.quack( ) ).then( ( ) => {
             dispatch( actions.swim( distance ) );
             dispatch( /* any action */ );
         });
11
12
13
     export default {
         simpleQuack,
14
         complexQuack
15
     };
```

```
File Edit Selection View Go Run Terminal Help
                                                                  • import { combineReducers } from "redux"; • Untitled-2 - wildrydes-site - Visual Studio Code
×1
                                                                                                                                                                                             ٥
                                                                                                Re-ducks - reducers
                                                                                                                                                                                              ···· 🖽
                      🛱 // widgets.js Untitled-1 🄍 🛛 🛱 import { combineReducers } from "redux"; Untitled-2 .
¢,
                                                                                                                                                                                         import { combineReducers } from "redux";
             import types from "./types";
Q
P
B
             const quackReducer = ( state = false, action ) => {
                 switch( action.type ) {
                     case types.QUACK: return true;
                     default: return state;
              Υ.
             R
             const distanceReducer = ( state = 0, action ) => {
                 switch( action.type ) {
                     case types.SWIM: return state + action.payload.distance;
                     default: return state;
             const reducer = combineReducers( {
                 quacking: quackReducer,
                 distance: distanceReducer
             });
             export default reducer;
8
```

```
Prmaster ↔ ⊗ 0 △ 0 Git Graph ♦
```

503

Re-ducks - selectors



• Together with the operations, the selectors are part of the public interface of a duck. The split between operations and selectors resembles the <u>CQRS</u> <u>pattern</u>.

 Selector functions take a slice of the application state and return some data based on that. They never introduce any changes to the application state.

Re-ducks - index

This file specifies what gets exported from the duck folder. It will:

- export as default the reducer function of the duck.
- export as named exports the selectors and the operations.
- export the types if they are needed in other ducks.

1	<pre>import reducer from "./reducers";</pre>
	import reducer from tyreducers;
2	
3	<pre>export { default as duckSelectors } from "./selectors";</pre>
4	<pre>export { default as duckOperations } from "./operations";</pre>
5	<pre>export { default as duckTypes } from "./types";</pre>
6	
7	export default reducer;

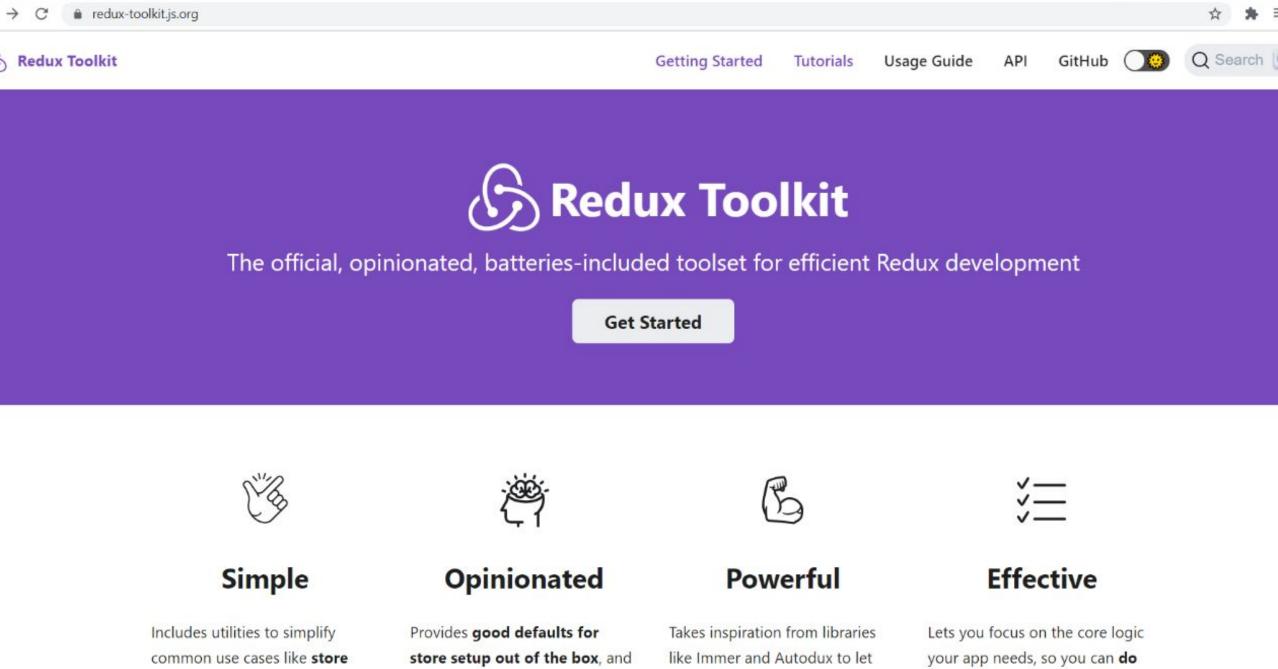
Ducks - tests

A benefit of using Redux and the ducks structure is that you can write your tests next to the code you are testing.

Testing your Redux code is fairly straight-forward:

Inside this file you can write tests for reducers, operations, selectors, etc.

```
import expect from "expect.js";
     import reducer from "./reducers";
     import actions from "./actions";
     describe( "duck reducer", function() {
         describe( "quack", function( ) {
             const quack = actions.quack( );
             const initialState = false;
             const result = reducer( initialState, quack );
10
11
             it( "should quack", function( ) {
12
                 expect( result ).to.be( true ) ;
13
14
             });
15
            );
16
```



setup, creating reducers, immutable update logic, and includes the most commonly used Redux addons built-in.

you write "mutative" immutable update logic. and more work with less code.

Redux Toolkit

- configureStore() wrapper for createStore. Default support redux-thunk and Redux DevTools Extension
- createReducer(): special syntax that lets you supply a lookup table of action types to case reducer functions, rather than writing switch statements. Uses Immer – so we can write "mutable" code for immutable updates
- createAction(): helper to create action creators
- createSlice(): helper to automatically generates a slice reducer with corresponding action creators and action types
- createAsyncThunk, createEntityAdapter, createSelector utility

Summary:

- ducks, re-ducks or redux-toolkit can use the same pattern/approach for all Redux code
- feature-based separation of the redux code is more flexible and allows more opportunities for scaling when codebase is growing
- Redux-toolkit provides useful tools and best practices
- How do you structure your redux apps?

Links:

- https://dev.to/alexandrudanpop/why-react-projects-still-use-redux-in-2020-395p
- <u>https://everyday.codes/javascript/architecture-of-large-react-apps-tools-and-techniques/</u>
- https://habr.com/ru/post/515700/
- <u>https://www.freecodecamp.org/news/scaling-your-redux-app-with-ducks-6115955638be/</u>
- https://github.com/erikras/ducks-modular-redux
- https://redux-toolkit.js.org/
- https://github.com/sergii-zhuravel/conf42-js2021

Thank you!

Sergii Zhuravel

szhuravell@gmail.com

https://twitter.com/SZhuravel