

CONF42

Ducks, Re-ducks, Redux-Toolkit

modular approaches for
React/Redux app structure

Sergii Zhuravel



Key facts

- **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 barely see how to compare those two

♥ 11 likes

💬 Reply

[Why React projects still use Redux](#)



Categories > Utilities > Client & Server Utilities > State Management

10 Best React State Management Libraries

Curated by the Openbase team and community. [Learn more](#)



xstate

[Save](#)

State machines and statecharts for the modern web.

License: MIT

TypeScript Definitions: Built-In

User Rating

5.0 / 5

12

Top Feedback

1 Great Documentation

GitHub Stars

18K

Weekly Downloads

848K

Last Commit

19hrs ago

Bundle Size (min+gzip)

N/A

Tree-Shakeable



redux

[Save](#)

Predictable state container for JavaScript apps

License: MIT

TypeScript Definitions: Built-In

User Rating

4.5 / 5

743

Top Feedback

54 Great Documentation

42 Performant

37 Highly Customizable

GitHub Stars

57K

Weekly Downloads

6M

Last Commit

3d ago

Bundle Size (min+gzip)

N/A

Tree-Shakeable

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

```
> node_modules
▼ src
  > actions
  > api
  > common
  > components
  > constants
  > containers
  > reducers
  > routes
  > sagas
  > selectors
  JS App.js
  <> index.html
  JS index.js
  B .babelrc
  .eslintrc.js
  .gitignore
  {} package.json
  {} package-lock.json
  README.md
  webpack.config.js
```

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	└─ cart
└─ JS cartActions.js	└─ JS cartActions.js
└─ JS productActions.js	└─ JS cartItem.js
└─ JS sessionActions.js	└─ JS cartReducers.js
└─ components	└─ JS shoppingCart.js
└─ JS cartItem.js	└─ common
└─ JS footer.js	└─ JS footer.js
└─ JS header.js	└─ JS header.js
└─ JS loginForm.js	└─ JS home.js
└─ JS productBox.js	└─ product
└─ JS productButtons.js	└─ JS productActions.js
└─ JS registerForm.js	└─ JS productBox.js
└─ containers	└─ JS productButtons.js
└─ JS home.js	└─ JS productDetails.js
└─ JS productDetails.js	└─ JS productList.js
└─ JS productList.js	└─ JS productReducers.js
└─ JS shoppingCart.js	└─ session
└─ reducers	└─ JS loginForm.js
└─ JS cartReducers.js	└─ JS registerForm.js
└─ JS productReducers.js	└─ JS sessionActions.js
└─ JS sessionReducers.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, reducers, 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*. Problems - mix 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**)

Ducks - example

```
1 // widgets.js
2
3 // Actions
4 const LOAD = 'my-app/widgets/LOAD';
5 const CREATE = 'my-app/widgets/CREATE';
6 const UPDATE = 'my-app/widgets/UPDATE';
7 const REMOVE = 'my-app/widgets/REMOVE';
8
9 // Reducer
10 export default function reducer(state = {}, action = {}) {
11   switch (action.type) {
12     // do reducer stuff
13     default: return state;
14   }
15 }
16
17 // Action Creators
18 export function loadWidgets() {
19   return { type: LOAD };
20 }
21
22 export function createWidget(widget) {
23   return { type: CREATE, widget };
24 }
25
26 export function updateWidget(widget) {
27   return { type: UPDATE, widget };
28 }
29
30 export function removeWidget(widget) {
31   return { type: REMOVE, widget };
32 }
33
34 // side effects, only as applicable
35 // e.g. thunks, epics, etc
36 export function getWidget () {
37   return dispatch => get('/widget').then(widget => dispatch(updateWidget(widget)))
38 }
```

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

Ducks - origin of the name

«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';
```

Ducks - use

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

- *React Redux Universal Hot Example* uses ducks. See */src/redux/modules*
- Todomvc using ducks

Meet Re-ducks

“The original ducks modular approach 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.

```
1  const QUACK = "app/duck/QUACK";
2  const SWIM = "app/duck/SWIM";
3
4  export default {
5    QUACK,
6    SWIM
7  };
```

Re-ducks -actions

This file contains all the action creator functions.

```
1 import types from "../types";
2
3 const quack = ( ) => ( {
4   | type: types.QUACK
5 } );
6
7 const swim = ( distance ) => ( {
8   | type: types.SWIM,
9   | payload: {
10  |   | distance
11  |   }
12 } );
13
14 export default {
15   | swim,
16   | quack
17 };
```


Re-ducks - operations

To represent chained operations you need a redux *middleware* to enhance the dispatch function. Some popular examples are: [redux-thunk](#), [redux-saga](#) or [redux-observable](#).

```
1  import actions from "./actions";
2
3  const simpleQuack = actions.quack;
4
5  // This is a thunk which dispatches multiple actions from actions.js
6  const complexQuack = ( distance ) => ( dispatch ) => {
7    dispatch( actions.quack( ) ).then( ( ) => {
8      dispatch( actions.swim( distance ) );
9      dispatch( /* any action */ );
10   } );
11 }
12
13 export default {
14   simpleQuack,
15   complexQuack
16 };
```

Re-ducks - reducers

index.html // widgets.js Untitled-1 • import { combineReducers } from "redux"; Untitled-2 •

```
1 import { combineReducers } from "redux";
2 import types from "./types";
3
4 /* State Shape
5 {
6   quacking: bool,
7   distance: number
8 }
9 */
10
11 const quackReducer = ( state = false, action ) => {
12   switch( action.type ) {
13     case types.QUACK: return true;
14     /* ... */
15     default: return state;
16   }
17 }
18
19 const distanceReducer = ( state = 0, action ) => {
20   switch( action.type ) {
21     case types.SWIM: return state + action.payload.distance;
22     /* ... */
23     default: return state;
24   }
25 }
26
27 const reducer = combineReducers( {
28   quacking: quackReducer,
29   distance: distanceReducer
30 } );
31
32 export default reducer;
```

Re-ducks - selectors

```
1  function checkIfDuckIsInRange( duck ) {  
2    |   return duck.distance > 1000;  
3  }  
4  
5  export default {  
6    |   checkIfDuckIsInRange  
7  };
```

- Together with the operations, the selectors are part of the public interface of a duck. The split between operations and selectors resembles the CQRS pattern.
- 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 import reducer from "./reducers";
2
3 export { default as duckSelectors } from "./selectors";
4 export { default as duckOperations } from "./operations";
5 export { default as duckTypes } from "./types";
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.

```
1 import expect from "expect.js";
2 import reducer from "../reducers";
3 import actions from "../actions";
4
5 describe( "duck reducer", function( ) {
6   describe( "quack", function( ) {
7     const quack = actions.quack( );
8     const initialState = false;
9
10    const result = reducer( initialState, quack );
11
12    it( "should quack", function( ) {
13      expect( result ).to.be( true );
14    } );
15  } );
16 } );
```



Redux Toolkit

The official, opinionated, batteries-included toolset for efficient Redux development

[Get Started](#)



Simple

Includes utilities to simplify common use cases like **store setup, creating reducers, immutable update logic**, and



Opinionated

Provides **good defaults for store setup out of the box**, and includes **the most commonly used Redux addons built-in**.



Powerful

Takes inspiration from libraries like Immer and Autodux to let you **write "mutative" immutable update logic**, and



Effective

Lets you focus on the core logic your app needs, so you can **do 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>
- <https://everyday.codes/javascript/architecture-of-large-react-apps-tools-and-techniques/>
- <https://habr.com/ru/post/515700/>
- <https://www.freecodecamp.org/news/scaling-your-redux-app-with-ducks-6115955638be/>
- <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>