Bottom of Form

Redux Observable RxJS: Going Epic with Reactive Programming



This React tutorial is part 2 of 2 in the series.

Part 1: [The SoundCloud Client in React + Redux](https://www.robinwieruch.de/the-soundcloud-client-in-react-redux/)

The good thing about the Redux + React ecosystem is you can always have a shot at something novel in the JavaScript landscape. Especially when it comes to the middleware of Redux (thunk) to handle asynchronous requests, one will find a great selection of paradigms and implementations to choose from. **In the beginning you often end up with a Promise based middleware to deal with asynchronous actions**, but there is more in JavaScript like Observables and Generators to deal with asynchronous requests.

The tutorial will show you how to use **Observables in Redux. Moreover it will give you an entry point into Reactive Programming**. Keep in mind that it will only scratch the surface, but maybe afterwards you get the idea and want to give it a shot in some of your projects as well.

Reactive Programming gained a lot of attraction in the recent years. Netflix is one of the bigger companies using and representing it openly. The [Rx family](http://reactivex.io/) is well known for providing libraries for all kinds of platforms to make Reactive Programming happen - JavaScript is one of these platforms.

The Redux Observable RxJS: Going Epic with Reactive Programming tutorial uses [**redux-observable**](https://github.com/redux-observable/redux-observable)**as middleware for asynchronous actions in Redux**. **It takes advantage of [RxJS](https://github.com/reactivex/rxjs" \t "_blank) to provide observable streams**. Observables are in contrast to Arrays collections over time - one can call them streams as well. Operators (filter, map, scan..) on Observables allow one to return a new stream while keeping the old stream untouched. That characteristic is called immutability, because we don’t alter the previous object. We might be already familiar with the same functionality (filter, map, reduce…) in plain JavaScript Arrays, but we can also apply them on streams over time. For instance one can even merge two streams into one stream to enable orchestration of the business logic.

Seed Project?

If you don’t have the outcome of [The SoundCloud Client in React + Redux](http://www.robinwieruch.de/the-soundcloud-client-in-react-redux/), checkout this [repository](https://github.com/rwieruch/react-redux-soundcloud). Clone the repository and use the Base folder named base-init/ as seed project. The seed project is the same which you would implement when you would step through the [initial tutorial](http://www.robinwieruch.de/the-soundcloud-client-in-react-redux/). Next to the Base folder you will find several Extension folders, which will show the outcome of each tutorial which builds up on The SoundCloud Client in React + Redux.

When you have your own [The SoundCloud Client in React + Redux](http://www.robinwieruch.de/the-soundcloud-client-in-react-redux/) project, you can continue with the [Table of Contents](https://www.robinwieruch.de/redux-observable-rxjs/#toc) and ignore the following instructions.

When you want to use the seed project, make sure to clone the [repository](https://github.com/rwieruch/react-redux-soundcloud). Use the base-init/ folder as your project folder. Make sure that your project starts with npm install && npm start.

Table of Contents

* [What is an Epic?](https://www.robinwieruch.de/redux-observable-rxjs/#epic)
* [Epics Middleware in Redux](https://www.robinwieruch.de/redux-observable-rxjs/#epicsMiddleware)
* [Troubleshoot](https://www.robinwieruch.de/redux-observable-rxjs/#troubleshoot)
* [Final Thoughts](https://www.robinwieruch.de/redux-observable-rxjs/#finalThoughts)

What is an Epic?

Let’s scratch the surface of Reactive Programming with the usage of redux-observable. First we install the [redux-observable](https://github.com/redux-observable/redux-observable) middleware module.

*From root folder:*

npm install --save redux-observable

Moreover redux-observable depends on Reactive Programming principles which are provided by [RxJS](https://github.com/reactivex/rxjs" \t "_blank).

*From root folder:*

npm install --save rxjs

We need to import rxjs explicitly to get all the operators (filter, map, scan..) on observables. The root file is sufficient for importing the module.

*src/index.js*

import 'rxjs';

...

After we’ve set up everything accordingly, we can start to replace the thunk approach observables. We do this from scratch and you can compare the solution afterwards with the thunk approach we used before.

First we need to authenticate our app with the SoundCloud API ([What’s an API?](https://www.robinwieruch.de/what-is-an-api-javascript/)).

*src/actions/auth.js*

import \* as actionTypes from '../constants/actionTypes';

export function auth() {

return {

type: actionTypes.AUTH

};

}

The authentication process gets started by dispatching the auth function. We already do this in the Stream container component. No payload is necessary in the action creator, since we are only activating the whole authentication procedure.

Another important point is that the action type will not be represented in the reducer, because it only activates the process to authenticate. We are used to have actions + reducer pairs, but in redux-observable we can abuse the Redux middleware to trigger a whole process with one single action without pairing it to a reducer. The process itself will trigger multiple actions which our reducer will respect to store data.

Additionally we need to add the new action type in our constants.

*src/constants/actionTypes.js*

export const AUTH = 'AUTH';

export const ME\_SET = 'ME\_SET';

export const TRACKS\_SET = 'TRACKS\_SET';

export const TRACK\_PLAY = 'TRACK\_PLAY';

Now the concept of Epics comes into play. The core primitive of redux-observable are Epics. An Epic is a function which takes a stream of actions and returns a stream of actions.

function (action$: Observable<Action>, store: Store): Observable<Action>;

Inside an Epic you can use the observable operators to create new streams or to orchestrate them. It is important to know that the outermost stream after all has to return an action for the Redux middleware.

Let’s setup our first Epic!

*src/actions/auth.js*

import SC from 'soundcloud';

import \* as actionTypes from '../constants/actionTypes';

import { Observable } from 'rxjs';

export function auth() {

return {

type: actionTypes.AUTH

};

}

export const authEpic = (action$) =>

action$.ofType(actionTypes.AUTH)

.mergeMap(() =>

Observable.from(SC.connect())

.map(setSession)

);

An Epic gives us an action$ observable as argument. The $ indicates that we are dealing with an observable. The observable has the helper function ofType to determine the type of the incoming action. Now we can listen to the AUTH action we triggered before.

Additionally we are using observable operators to chain a bunch of streams. Sc.connect initializes the connection to SoundCloud and returns a Promise which gets eventually resolved and returns a session object. We use Observable.from to turn a Promise into an Observable. Afterwards we would be able to catch errors on the stream:

Observable.from(SC.connect())

.map(setSession)

.catch(setSessionError);

Once the login of the SC.connect succeeds and the Promise resolves, it returns a session object. The output of a stream is the input of the next stream in the chain. Therefore we know that we get the session object to call setSession in shorthand which returns an action at the end of the Epic.

Let’s explain shortly the operators we used.

* map - It maps one stream to another stream in a synchronous way.
* mergeMap - It maps one stream to another stream in an asynchronous way. Commonly it is used to handle asynchronous requests. In non Reactive Programming map is used to map from a synchronous object to another synchronous object. One can use mergeMap to map from a synchronous object to an asynchronous object. For instance it can be used to map from an URL string to a Promise based HTTP request which gets resolved eventually.

But we didn’t implement setSession yet!

*src/actions/auth.js*

import SC from 'soundcloud';

import \* as actionTypes from '../constants/actionTypes';

import { Observable } from 'rxjs';

export function auth() {

return {

type: actionTypes.AUTH

};

}

function setSession(session) {

return {

type: actionTypes.SESSION\_SET,

session

};

}

export const authEpic = (action$) =>

action$.ofType(actionTypes.AUTH)

.mergeMap(() =>

Observable.from(SC.connect())

.map(setSession)

);

Again we need to add the action type in our constants.

*src/constants/actionTypes.js*

export const AUTH = 'AUTH';

export const SESSION\_SET = 'SESSION\_SET';

export const ME\_SET = 'ME\_SET';

export const TRACKS\_SET = 'TRACKS\_SET';

export const TRACK\_PLAY = 'TRACK\_PLAY';

Now the authentication is finished. Let’s recap our first Epic driven process. We dispatch an action with a plain action creator. No reducer is responsible for the action type, but it kicks off our Epic. The Epic encapsulates the whole authentication process. At the end the Epic returns an action to set the session.

There could be a reducer listening to the action, but we don’t need the session object for the sake of simplicity in the global state. But feel free to save the object in the global state on your own! Moreover we could implement error handling as well.

As we learned we can use actions to trigger Epics. Let’s use the last (unused) action to trigger two simultaneous Epics! One Epic to retrieve the user object and one epic to retrieve the list of tracks of the user. Both requests only need the session object, which we already send in the payload of the last action.

*src/actions/auth.js*

...

export const fetchMeEpic = (action$) =>

action$.ofType(actionTypes.SESSION\_SET)

.mergeMap((action) =>

Observable.from(fetchMe(action.session))

...

);

export const fetchStreamEpic = (action$) =>

action$.ofType(actionTypes.SESSION\_SET)

.mergeMap((action) =>

Observable.from(fetchStream(action.session))

...

);

As you can see both Epics listen to the SESSION\_SET action type. Afterwards we use again mergeMap and Observable.from to handle the API requests. We didn’t implement both functions fetchMe and fetchSteam yet. Let’s implement them.

*src/actions/auth.js*

...

export const fetchMeEpic = (action$) =>

action$.ofType(actionTypes.SESSION\_SET)

.mergeMap((action) =>

Observable.from(fetchMe(action.session))

...

);

export const fetchStreamEpic = (action$) =>

action$.ofType(actionTypes.SESSION\_SET)

.mergeMap((action) =>

Observable.from(fetchStream(action.session))

...

);

const fetchMe = (session) =>

fetch(`//[api.soundcloud.com/me?oauth\_token=](http://api.soundcloud.com/me?oauth_token=)${session.oauth\_token}`)

.then((response) => response.json());

const fetchStream = (session) =>

fetch(`//[api.soundcloud.com/me/activities?limit=20&offset=0&oauth\_token=](http://api.soundcloud.com/me/activities?limit=20&offset=0&oauth_token=)${session.oauth\_token}`)

.then((response) => response.json());

The requests should work now. Let’s fire actions to set the data in the global state object.

*src/actions/auth.js*

import { setTracks } from '../actions/track';

...

function setMe(user) {

return {

type: actionTypes.ME\_SET,

user

};

}

...

export const fetchMeEpic = (action$) =>

action$.ofType(actionTypes.SESSION\_SET)

.mergeMap((action) =>

Observable.from(fetchMe(action.session))

.map(setMe)

);

export const fetchStreamEpic = (action$) =>

action$.ofType(actionTypes.SESSION\_SET)

.mergeMap((action) =>

Observable.from(fetchStream(action.session))

.map((data) => setTracks(data.collection))

);

const fetchMe = (session) =>

fetch(`//[api.soundcloud.com/me?oauth\_token=](http://api.soundcloud.com/me?oauth_token=)${session.oauth\_token}`)

.then((response) => response.json());

const fetchStream = (session) =>

fetch(`//[api.soundcloud.com/me/activities?limit=20&offset=0&oauth\_token=](http://api.soundcloud.com/me/activities?limit=20&offset=0&oauth_token=)${session.oauth\_token}`)

.then((response) => response.json());

We can reuse the action creator we already have in place to set the tracks from a different file. We don’t have to refactor the *src/actions/track.js* file, because it has only plain action creators. The whole file should look like the following.

*src/actions/auth.js*

import SC from 'soundcloud';

import \* as actionTypes from '../constants/actionTypes';

import { setTracks } from '../actions/track';

import { Observable } from 'rxjs';

export function auth() {

return {

type: actionTypes.AUTH

};

}

function setSession(session) {

return {

type: actionTypes.SESSION\_SET,

session

};

}

function setMe(user) {

return {

type: actionTypes.ME\_SET,

user

};

}

export const authEpic = (action$) =>

action$.ofType(actionTypes.AUTH)

.mergeMap(() =>

Observable.from(SC.connect())

.map(setSession)

);

export const fetchMeEpic = (action$) =>

action$.ofType(actionTypes.SESSION\_SET)

.mergeMap((action) =>

Observable.from(fetchMe(action.session))

.map(setMe)

);

export const fetchStreamEpic = (action$) =>

action$.ofType(actionTypes.SESSION\_SET)

.mergeMap((action) =>

Observable.from(fetchStream(action.session))

.map((data) => setTracks(data.collection))

);

const fetchMe = (session) =>

fetch(`//[api.soundcloud.com/me?oauth\_token=](http://api.soundcloud.com/me?oauth_token=)${session.oauth\_token}`)

.then((response) => response.json());

const fetchStream = (session) =>

fetch(`//[api.soundcloud.com/me/activities?limit=20&offset=0&oauth\_token=](http://api.soundcloud.com/me/activities?limit=20&offset=0&oauth_token=)${session.oauth\_token}`)

.then((response) => response.json());

Epics Middleware in Redux

Now we know the concept around Epics, but someone has to introduce them to the Redux store. The redux-observables module comes with a middleware function, which takes all combined Epics as argument. Afterwards the created middleware can be used to create the overall store middleware.

*src/stores/configureStore.js*

import { createStore, applyMiddleware } from 'redux';

import createLogger from 'redux-logger';

import { browserHistory } from 'react-router';

import { createEpicMiddleware } from 'redux-observable';

import { routerMiddleware } from 'react-router-redux';

import { rootEpic } from '../actions/index';

import rootReducer from '../reducers/index';

const logger = createLogger();

const router = routerMiddleware(browserHistory);

const epicMiddleware = createEpicMiddleware(rootEpic);

const createStoreWithMiddleware = applyMiddleware(epicMiddleware, router)(createStore);

export default function configureStore(initialState) {

return createStoreWithMiddleware(rootReducer, initialState);

}

But one ingredient is missing: the combined rootEpic. Like combineReducer for multiple reducers in Redux, we can use the combineEpics helper from redux-observable to export a rootEpic for the Epic middleware. We simply import all Epics in our root actions file and export them as a combined rootEpic.

*src/actions/index.js*

import { combineEpics } from 'redux-observable';

import { auth, authEpic, fetchMeEpic, fetchStreamEpic } from './auth';

import { setTracks, playTrack } from './track';

const rootEpic = combineEpics(

authEpic,

fetchMeEpic,

fetchStreamEpic

);

export {

auth,

setTracks,

playTrack,

rootEpic

};

The app should work again.

*From root folder:*

npm start

As I said in the beginning, the world around Reactive Programming is so much bigger. The ecosystem of React and Redux allows one to experiment with a lot of different paradigms. The redux-observable middleware is only a starting point for Reactive Programming. If you want to dive deeper into Reactive Programming, I can recommend the [egghead.io](https://egghead.io/) courses about the topic and a [great read by Andre Staltz](https://gist.github.com/staltz/868e7e9bc2a7b8c1f754).

Troubleshoot

You may encounter issues in that tutorial. Here you will find some references how to handle issues.

**Dependencies**

In case you want to know which versions npm installed during that tutorial, here a list of all npm packages in my package.json.

"devDependencies": {

"babel-core": "^6.9.1",

"babel-loader": "^6.2.4",

"babel-preset-es2015": "^6.9.0",

"babel-preset-react": "^6.5.0",

"babel-preset-stage-2": "^6.5.0",

"chai": "^3.5.0",

"enzyme": "^2.3.0",

"exports-loader": "^0.6.3",

"imports-loader": "^0.6.5",

"jsdom": "^9.2.1",

"mocha": "^2.5.3",

"react-addons-test-utils": "^15.1.0",

"react-hot-loader": "^1.3.0",

"webpack": "^1.13.1",

"webpack-dev-server": "^1.14.1"

},

"dependencies": {

"react": "^15.1.0",

"react-dom": "^15.1.0",

"react-redux": "^4.4.5",

"react-router": "^2.4.1",

"react-router-redux": "^4.0.5",

"redux": "^3.5.2",

"redux-logger": "^2.6.1",

"redux-observable": "^0.9.0",

"rxjs": "^5.0.0-beta.11",

"soundcloud": "^3.1.2",

"whatwg-fetch": "^1.0.0"

}

Final Thoughts

Have a look again in [The SoundCloud Client in React + Redux](http://www.robinwieruch.de/the-soundcloud-client-in-react-redux/) tutorial for more smaller tutorials which build up on top of the SoundCloud project. All tutorials on top will get released in [the repository](https://github.com/rwieruch/react-redux-soundcloud) as well.

I would like to hear your thoughts :-) Find me on [Twitter](https://twitter.com/rwieruch) and [GitHub](https://github.com/rwieruch)

Did the article help you? You can share it with your friends on social media , support me on [Patreon](https://www.patreon.com/rwieruch" \t "_blank) or take one of my courses

The Road to learn React



Build a Hacker News App along the way. No setup configuration. No tooling. No Redux. Plain React in 190+ pages of learning material. Learn React like **21.000+ readers**.

[**GET THE BOOK**](https://www.getrevue.co/profile/rwieruch)

Never miss an article about web development and self-growth.

TAKE PART

 Join 11.000+ Developers

 Learn Web Development with JavaScript

 Tips and Tricks

 Access Tutorials, eBooks and Courses

 Personal Development as a Software Engineer

Top of Form

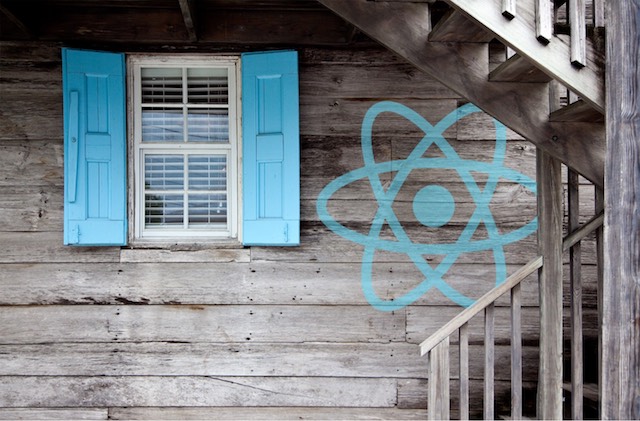
SUBSCRIBE

Bottom of Form

**RECENT POSTS**

[](https://www.robinwieruch.de/react-testing-tutorial/)

[**REACT TESTING TUTORIAL: TEST FRAMEWORKS & COMPONENT TESTS**](https://www.robinwieruch.de/react-testing-tutorial/)

[](https://www.robinwieruch.de/react-js-windows-setup/)

[**A SIMPLE REACT.JS ON WINDOWS SETUP**](https://www.robinwieruch.de/react-js-windows-setup/)

[](https://www.robinwieruch.de/the-road-to-learn-react/)

[**THE ROAD TO LEARN REACT [REACT.JS BOOK FOR 2018]**](https://www.robinwieruch.de/the-road-to-learn-react/)

**ABOUT ME**



Hi, I'm Robin - a german and Berlin based software engineer. You can find me on [Twitter](https://twitter.com/rwieruch), [GitHub](https://github.com/rwieruch) and [Patreon](https://www.patreon.com/rwieruch). Find out more [about me](https://www.robinwieruch.de/about) or [work with me!](https://www.robinwieruch.de/work-with-me)

* © Robin Wieruch

* [Legal Notice](https://www.robinwieruch.de/legal)