MODERN REDUX

Learn redux toolkit and the new redux best practices

- OHANS EMMANUEL -

Table of Contents

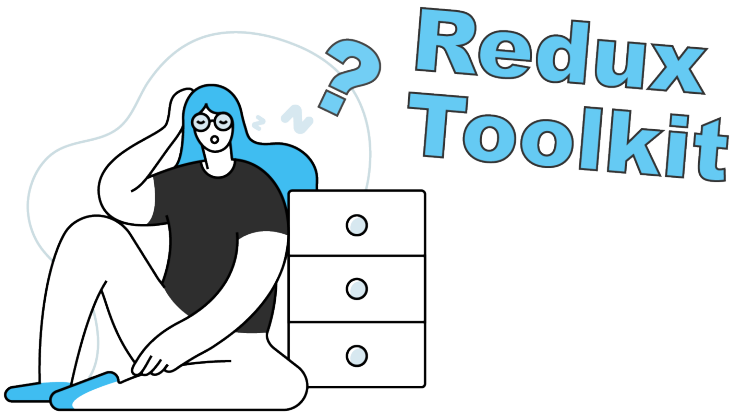
- Syncing UI state with the Redux Store
- Initial Application Data Fetch
- Thunks and Redux Toolkit
- Thunk Dispatch and handling the UI loading state
- Async Error Handling Logic in Thunks
- How to log draft state values
- Simplifying Thunks with createAsyncThunk
- Using Promise Lifecycle Actions from createAsyncThunk
- Refactoring Tweetfind to use createAsyncThunk
- Completing Tweetfind

Chapter 4: Where to go from here

- The best practice on data fetching
- Redux-toolkit with Typescript
- Testing a modern Redux application

Conclusion

Section 1: What is Redux Toolkit, and why use it?



Hey! Welcome to the modern redux book! If you can sense excitement through text, then I hope these sentences are all popping out at you!

Before I get you started on the meat of the book, it's important to make sure this book is right for you. There's no use studying halfway through the book only to find out, like a bad first day at a new job, it's not what you expected.

Let's set expectations straight. Only then can you get the best out of this book – and that's my goal.

Who is this book for?

This book has been written for 2 categories of people. Sorry to remind you of Biology, but do you remember the stages a butterfly undergoes, i.e. from egg to caterpillar… ?

Well, here are the categories of people this book has been written for.

1. The Caterpillars

You fall under this category if:

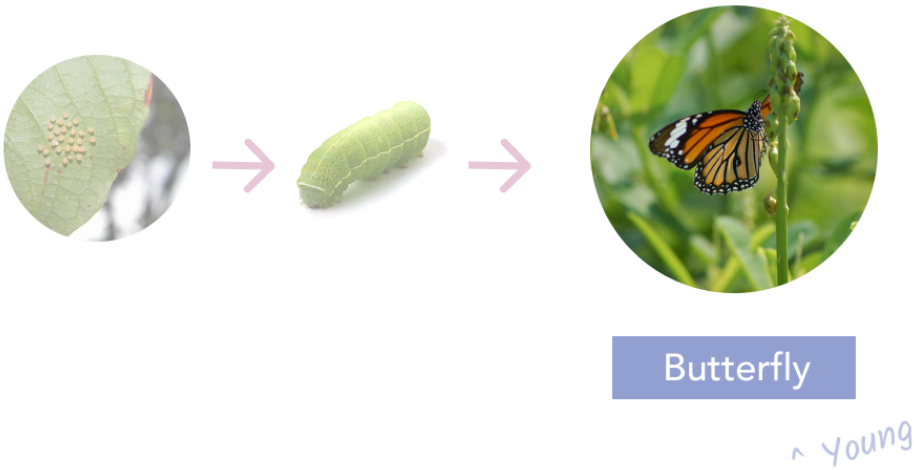
- You have already learned the basics of Redux.
- The fundamental concepts of actions, reducers and the Redux store aren't new to you.

Good news Caterpillar, I've got you covered. This book is a logical next step if you already understand the basics of Redux. I'll introduce concepts that build upon that

foundational knowledge you've got and show you how modern Redux should be written with the Redux toolkit.

Caterpillars don't fly. But that's ok! As you work through the book, your pace may be slower than that of someone who's been writing Redux for more years than you have. Don't stress over that. I'll do my best to explain every detail in very approachable language. I just need you to hang in there, Caterpillar. Don't try to fly. Easy. Slow and steady, and soon enough you'd be flying high with beautiful fluttered wings.

2. The Young Butterfly



So, you're a young butterfly, eh? Not sure? That's okay.

You fall under this category if:

- You are experienced with Redux but are new to Redux toolkit.
- You're unfamiliar with modern Redux and need a refresher

I've got you covered young Butterfly. I was in these shoes sometime back, so I definitely understand and can relate to your needs. The frontend world moves really fast.

In this book, I'll show you the much needed RTK introduction and refresher that you need.

Since you're no beginner to Redux, I reckon you'd be able to assimilate the concepts and usage of RTK a lot faster than a beginner. That's superb.

However, try your hands on the book challenges. They'll strengthen your understanding of RTK and help improve your Redux development in general.

What this book isn't

There are two things this book isn't.

In a nutshell:

- This book is not a *"... teach me the absolute basics of Redux book"*. Careful, Caterpillar. I strongly recommend my Understanding Redux #1 book for the basics of Redux. With a 5-star rating (247 ratings in total), you'll definitely be in good hands.

- This book is not a *"... teach me every single thing RTK has to offer book"*. That's why a technical documentation exists. Yes, this book covers the basics and intermediate uses of RTK, and even some advanced concepts, however certain advanced concepts have been intentionally left out. After going through this book, I have no doubts you'd easily pick those up from the official RTK documentation as this book sets you up on the right foundation.

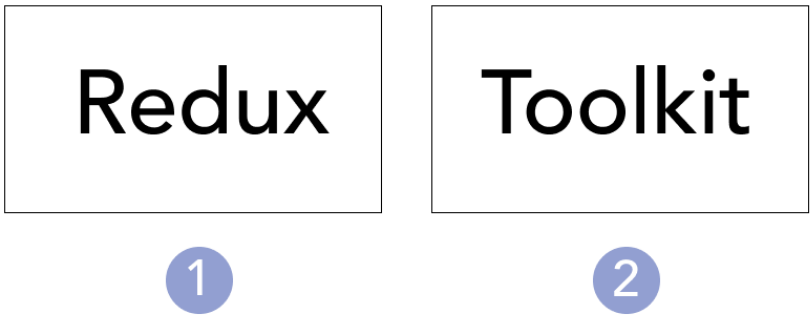
Okay, enough of the housekeeping. Let's get started crunching Redux toolkit from the next lesson. Yay.

What is Redux Toolkit (RTK)?

If we're going to be spending the next moments of life discussing RTK, it's only fair to start off by explaining what Redux toolkit is.

So, here we go.

Let's break the library's monicker, i.e., *Redux toolkit*, apart. That's a decent place to start.



We get the words "Redux" and "toolkit". You already know what Redux is. A toolkit, you probably already know what that means. In a nutshell, a toolkit refers to a set of tools. It's that simple, like a toolbox.

When you bring both words together, it gives a clue into what RTK really is. A toolbox for handling common redux use cases.

While tools like screwdrivers and hammers do a lot of good in the real world, they aren't quite useful with software. Except you want to smash your monitor with a hammer, then certainly, be my guest.

If Redux toolkit isn't issuing out hammers and screwdrivers, then what does the toolbox comprise? Take a guess.

If you said utilities, you're right. Think of utility functions exported from the library, where these functions exist to make redux development a lot more fun. Less boilerplate, so you can really focus on being efficient writing code. This is the entire premise of Redux toolkit.

While the above statements describe the absolute core of RTK, there's one more thing to note.

When someone hands you a tool to do a job, in a sense, they've restricted your options. If I handed you a screwdriver to perform an experiment, I'm indirectly saying to you, this tool solves the problem. Just use it, but how you do is up to you.

What I'm trying to say is, when the Redux core maintainers say, Redux Toolkit is the tool of choice for modern Redux development, they've handed everyone a tool.

Under the hood, they've also made numerous opinionated decisions on how the tool works. The utility functions work in a certain way.

This leads to RTK being opinionated. Out of the box, it makes choices for the Redux store setup and includes many redux add-ons. I'll point out many of these defaults, so you understand them. If anytime in your Redux development you become uncomfortable with any of the defaults, you'd be able to go make the needed changes.

Finally, tools exist to solve certain problems. With Redux toolkit, it helps simplify common use cases such as setting up the redux store, creating reducers and handling immutable logic within your reducers.

In the next lesson, I'll elaborate on the problems RTK tries to solve. When I started using RTK, I asked myself why the Redux maintainers waited years before issuing this much-needed toolbox. See you in the next lesson!

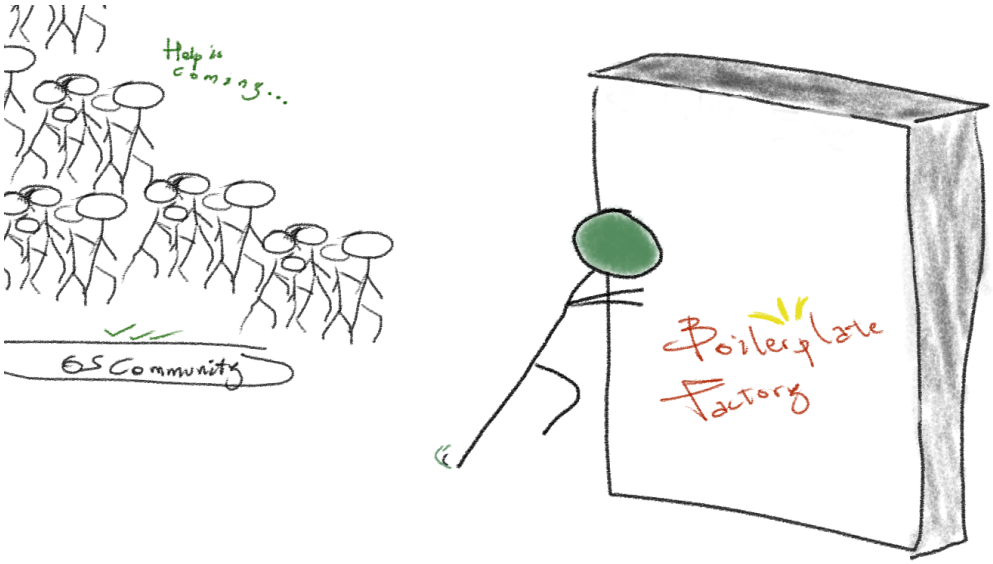
The problems Redux toolkit tries to solve

The problems RTK attempts to solve become apparent as soon as you start building redux apps with RTK. Let's look at the number one culprit.

The number one complaint I get from redux users is how much boilerplate code they write with Redux. "Redux requires too much boilerplate code". I heard and read that too many times, but understandably so.

As developers, we are focused on solving problems efficiently. Spending unnecessary time writing boilerplate code is far from efficient. Like an uncomfortable itch, it's hard to live with wiring boilerplates all the time.

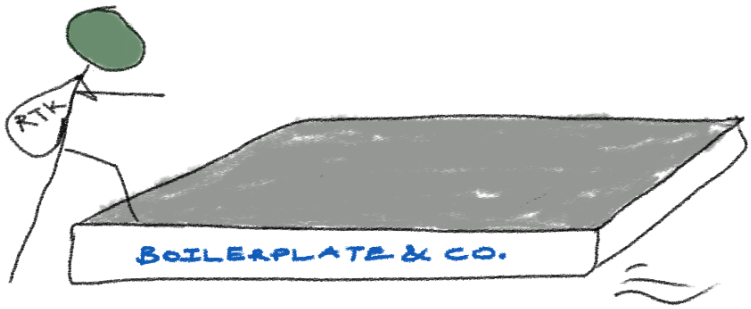
Even though this book is about RTK, the truth is before RTK there have been many open-source libraries created to help take away some pain associated with redux — writing boilerplate code being the lead character!



However, with RTK you get a solution from the official maintainers of Redux. RTK is touted as **the intended way to write modern Redux logic**.

More than just reducing boilerplate code, RTK aims to address other problems such as the **difficulty in creating reducers**, creating a store with multiple actors. Creating string constants, endless switch cases – this can all be a hassle.

Well, look no further. Redux toolkit addresses these very common problems.



In the spirit of fairness, I have to mention that no one drug cures all pains. So, Redux toolkit may not solve all of yours. It is limited in scope, but it definitely kicks a lot of the aforementioned problems in the butt. Concerns such as folder structure and data caching would still have to be addressed by you – no running away from caching, eh? ;)

However, I'll discuss some best practices as we build example applications in the book.

What's included in Redux Toolkit?

Remember I said RTK was a toolset comprising utility functions to make redux development a lot easier. The rest of the book is dedicated to taking a deep look at these utility functions, but let's take a quick peep into them.

The Redux Toolkit Main dependencies

To be completely honest you could skip this section and move on to the next. That's more important if you care about the exposed APIs redux toolkit provides.

Okay, for the nerds, let's talk about the inner libraries employed by redux-toolkit. For RTK to deliver its promises of a better Redux development, it also employs help from some reputable open-source libraries.

You don't need a degree to know what the main dependencies are. You just need a package.json.

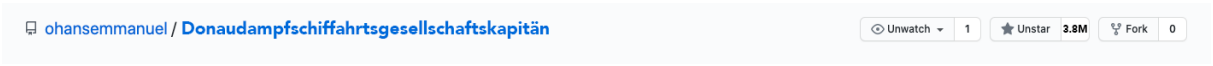
At the time of this writing, RTK has four main dependencies:

```
// 1. immer
// 2. redux
// 3. redux-thunk
// 4. reselect
```

If you've written Redux in the past, you likely recognise all or a few of those. Come on, you recognise the second dependency!

Immer and Redux Toolkit

The most interesting of the dependencies is arguably *Immer*, the German word for "always" in English. Goodness! Imagine if that was a longer German word like *Donaudampfschiffahrtsgesellschaftskapitän*. Then, I guess we'd all star the library to remember its name (try this novel trick to steal GitHub stars)



Other than the impressive name, it was named the breakthrough of the year on the open-source award. So, what does Immer do?

When you first learned redux, you remember the rule "do not mutate state in your reducers"?

Consequently, you probably remember writing a lot of spread syntax magic like the following:

```
{
    ...state,
```

```
        [userId]: {
         ...allUserMsgs,
        }
}
```

How could you forget!

Well, Immer lets you throw that rule out the door (sort of). You write your typical mutable code, and it produces a new copy of state. Mutable code, immutable results.

```
// you can do this in your reducer
state.newValue = 5
```

RTK lets you do the same as it implements Immer. The good part is that you don't really need to learn about the inner workings of Immer to use RTK. Just go ahead and mutate your application state like no one cares. Well, RTK doesn't care. We will see practical examples of this in a bit!

Reselect and Redux Toolkit

Reselect is perhaps no news to an avid redux developer. However, what's interesting is that redux toolkit exports the `createSelector` utility from `reselect`. In all fairness, this is perhaps the function that gives you 80% of the results when you use `reselect`. Think Pareto principle.

Essentially, you don't have to reinstall `Reselect` to use `createSelector`. Just go ahead and import from the redux toolkit.

```
import { createSelector } from "@reduxjs/toolkit";
```

In the first real application I built with Redux toolkit, I went on to still add `reselect` as a dependency when all I used was the `createSelector` utility! That's how you know a redux toolkit rookie. You'll spot them all around. Don't say I told you.

The APIs exposed by Redux toolkit

The toolkit would be nothing if it didn't expose an intuitive (and helpful) API. Here are the APIs included. They give you a clearer sense of the problems RTK tries to solve.

- `configureStore`: Think of this as the good old `createStore` method with simplified configuration options and good defaults e.g., implements `redux-thunk` by default and enables the use of the Redux Devtools Extension by default.

- `createReducer`: This one is a lifesaver. Instead of writing switch statements, you supply a lookup table mapping action creators to reducer functions. Oh, and you can mutate state as much as you want thanks to `Immer`.

- `createAction`: generates an action creator function, so you don't have to do it. The function also has a `toString` method defined, so the function can be used in place of a `type` string. Pretty handy!

- `createSlice`: This is one of the most loved APIs I reckon. It lets you define your entire application state in slices. You pass an object of reducer functions, a slice name, an initial state value, and it automatically generates a slice reducer, action creators and action types! Talk about the use of the least effort!

- `createAsyncThunk`: This helps you create a thunk easily. The thunk will dispatch `pending`, `fulfilled` or `rejected` action types based on the promise returned.

- `createEntityAdapter`: This is great for managing normalised data in your redux store. It generates prebuilt reducers and selectors for performing CRUD operations on a normalised state structure.

- `createSelector`: Well, I talked about this earlier. It exists to spot `redux-toolkit` newbies. More seriously, it exports the function from `reselect`, so you don't have to worry about it.

Conclusion

Don’t worry if you didn’t understand the APIs above. I’ll go in details from the next section.

On the bright side, come on. Aren’t these amazing? I know Redux has been under a lot of criticism in the past, but this is a great direction for arguably the most mature client-side state management libraries out there.