Intro to Redux-ORM

We now have the initial app structure and UI layout set up properly, and we can start thinking about how to build the rest of the application. However, before we begin building, we need to take a detour and introduce a tool called **Redux-ORM**, which we'll be using heavily throughout the rest of our development.

Redux-ORM helps solve a number of use cases that are common to many Redux applications, particularly related to managing normalized relational data in your store. I've used it heavily in my own application, and have come up with some useful techniques and approaches for using it. Hopefully you'll find them useful in your own application as well.

Why Use Redux-ORM?

Client-side applications frequently need to deal with data that is nested or relational in nature. The standard advice for a Redux application is to store this data in a "normalized" form. For a Redux app, that means organizing part of your store to look like a set of database tables. Each type of item that you want to store gets an object that is used as a lookup table by mapping item IDs to item entries. Since objects don't have a real sense of order, arrays of item IDs are stored to indicate ordering.

> **Note**: For further information on normalization in Redux, see the Structuring Reducers section of the Redux docs, particularly the pages on Normalizing State Shape and Updating Normalized Data.

Because data is often received from the server in nested form, it needs to be transformed into a normalized form to be properly added to the store. The typical approach is to use the Normalizr library for this. You can define schema objects and how they relate, pass the root schema and some nested

data to Normalizr, and it gives you back a normalized version of the data suitable for merging into your state.

However, Normalizr is really only intended for one-time processing of incoming data, and only provides limited tools for dealing with normalized data once it's in your store. Normalizr 3.x does now include a function for denormalizing values back into a nested form, but it doesn't have functionality for updating relational items in the store.

There are also many other existing Redux addons for managing entities and relational data. Some of them will help fetch data from the server based on REST endpoints or JSON-API structures, others just provide prebuilt actions and reducers for managing data in the store.

Redux-ORM is a client-side only solution for defining relations between model classes, and acts as an abstraction layer for managing normalized relational values in your Redux store. It does not provide any data fetching capabilities on its own. It provides functions for selecting items and their relations from the store, and creating/updating entries while correctly implementing immutable updates internally so that you don't have to worry about immutability yourself.

My History With Redux-ORM

I came across Redux-ORM early in its development, as I was still learning Redux and just starting to build my first React+Redux app prototype. Redux-ORM fit my use cases very well, and Tommi Kaikkonen, the author, was very responsive and helpful. He answered the many questions I threw at him, fixed bugs as I reported them, and added some features that I asked for.

Since then, Redux-ORM has become a vital part of my toolkit for writing Redux apps. The data I'm working with is very nested and relational, and Redux-ORM has made dealing with that data a lot easier. Although it's not yet marked as version 1.0, the API has remained consistent and stable since its inception. The fact that the library actually comes with real meaningful documentation (both tutorials and API docs) is a huge plus as well.

I should also note that in early 2017, Tommi announced that he didn't have enough time to continue maintaining Redux-ORM, and put out a call for new maintainers. Several people volunteered to be "co-maintainers" or contributors, including myself. So, while I don't spend time working on

Redux-ORM specifically, I keep an eye on issues and discussion, and the library is still actively maintained.

Redux-ORM and Idiomatic Redux

There's been numerous addon libraries people have built that try to put some kind of OOP layer on top of Redux, as demonstrated by the "Variations" page in my Redux addons catalog. I've frequently pointed out that Redux is primarily focused on Functional Programming principles, and that OOP wrappers over Redux aren't idiomatic. So, given that I usually advise against using those sorts of libraries, you might ask why I encourage the use of Redux-ORM. What makes it different from other libraries like Jumpsuit or Radical?

Most of the OOP wrappers I've seen try to abstract things away by defining action creators as class methods, and often wind up ignoring the idea of multiple reducers being able to respond to a given action (or even making it impossible). **They treat Redux as something that needs to be hidden**, and end up throwing away many of the concepts that make Redux attractive.

On the other hand, **Redux-ORM doesn't try to hide Redux**. It doesn't pretend that action constants don't exist, or that actions and reducers are always a 1:1 correspondence. It ultimately just provides an abstraction layer over something you would otherwise would have written yourself: CRUD operations for normalized data. It enables me to think a little less about "What specific steps do I need to follow to update or retrieve this data properly?", and a little more about handling my data at a conceptual level.

Overall, **I highly recommend the use of Redux-ORM in any Redux app that needs to handle normalized nested/relational data**. It won't magically keep you from having to think about managing that data, but it *will* make it easier for you to deal with.

With that in mind, let's go take a look at how to use Redux-ORM.