Microblog

Springboard

It's time to build a blogging app that will eventually tie together your learning about React, Router, and Redux, along with full-stack development with a backend. Are you excited? (**Hint:** Of course you are!)

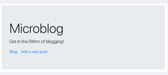
Two things this exercise is **not** about:

- CSS or design. There are lots of things with React & Redux to work on here, so go with the simplest layout/theme you can.
- Authentication: any site visitor will be able to do any of the functionality listed.

Part I: Basic Blog Functionality

This first part will **not use Redux** and **will not use a backend**. You should use React state and React Router.

Make a blogging site with the following features:



A site header/navigation box that appears on all pages, with links to the homepage ("Blog") and a post-add form ("Add a new post")

(We don't always show the header on mockups, but this should always appear on all pages).

A "new post" form that lets you enter a title, description, and body. This should be routed to */new*.

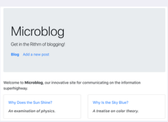
Canceling should redirect to the homepage.

Saving should also redirect to the homepage.

A "post view" page that shows a post. This should be routed to */[postId]*.

It should have a button that shows an edit form for the post; this does not need to be routed differently (the url should stay the same). The edit form can look like the new-post form, above.

It should have a button to delete the post. This should redirect to the homepage on deletion.



A "homepage list" that shows the title and description of each post. The title should be a link to the detail of the post. This should be routed to */*.

Note: Think About Your Components!

It's quite possible you might want the blog title/description list in other places than the homepage. It might make sense to have a simple homepage component, and a "title list" component with the logic for showing the list of titles.

Similarly, you should be able to use the same form for blog-adding and blog-editing.

Part II: Adding Comments

Add a feature that lets visitors comment on individual posts.

On the post-view page, it should list all existing comments (in the order they were made), and have a "delete" button (the red X in our mockup) to delete a comment.

Below that should be an inline form for adding a new comment.

Part III: Redux!

Convert your app to Redux. **For now, you still won't use a backend**.

Take some time to think about your Redux plan:

- It may be much easier and efficient to store the blog posts as an object, keyed by post ID, rather than an Array — this will make it easier for your reducers to edit/delete/find the right post, rather than always having to use *.find*, *.map*, *.filter*, etc.
- Figure out which components should have access to the redux store via *useSelector*

Make a plan **before** you start coding. Draw out what your Redux state would look like, and if your component hierarchy should change.

Once you've made your plan, you can compare it to our hierarchy:

Hover to reveal

Part IV: Add Our Backend

Download the backend.

We've provided an Express backend server with a simple API. You should look in the routes to see what the various routes it provides are.

Note: Ignore "votes" for now

A later part of this exercise adds voting capabilities, and our backend provides routes and data to support that — you can ignore this for now.

To use our backend, you'll need to create a database, ***microblog***. We provide an SQL file with the schema and a few sample posts/comments.

Change your app so that:

- it loads the simple list of id/title/description from the server for all posts, and uses this to show the list on the homepage
- when visiting a post, it retrieves the full detail of that post
- adding/editing/deleting posts updates the backend
- adding/deleting comments updates the backend

Of course, to do this well, you'll need to add ***redux-thunk***, so you can have asychronous actions that both deal with AJAX calls and then dispatch ordinary actions with the data returned.

Note: Redux-Thunk and the Dev Tools

When you add Redux-Thunk, you have to change the way you incorporate the dev tools into your store.

A nice way to do this is to add a simple NPM library, ***redux-devtools-extension***, which makes this a bit easier. Install that, and then change your ***index.js*** to include this:

index.js

```
import { createStore, applyMiddleware } from "redux";
import thunk from "redux-thunk";
import rootReducer from "./reducers/root"; // or wherever you put it
import { composeWithDevTools } from 'redux-devtools-extension';

const store = createStore(
  rootReducer,
  composeWithDevTools(
    applyMiddleware(thunk)));
```

Take your time and be thoughtful—this is the longest part of this exercise.

You find it helpful to rethink your Redux state; at this point, you might have been keeping just a simple object of all post data, and using that for both the title list and the post details. However, our server doesn't return *all* data at once (in a real site, this might be an enormous amount of data!).

So, you may find it helpful for your Redux state to have two different top-level keys:

- ***posts***: an object mapping postId: `{id, title, description, body, comments}`
- ***titles***: an array of the simple `{id, title, description}` you get from the backend when you get basic data on all posts

Writing the reducers here will take some thought—adding a new post, for example, will now need to update both the ***posts*** info *and* add to the ***titles*** info.

Bonus: it would be extra-nice if you kept in memory the post detail data for posts you've viewed, so you don't reload it if you revisit the post detail page for a post. So: load it from backend if you don't have it, re-use what you have in Redux if you do.

Part V: Voting

Add the last main feature: a post can be voted-up or voted-down. You should be able to do this on the homepage or the post-detail page, as shown:

In addition, the homepage should show the posts *most popular first* and, of course, as you up/down-vote posts, this may need to re-arrange.

Further Study

More Redux Features

You could experiment with Redux features like ***combineReducers***, which would let you have separate reducers for the ***posts*** and ***titles*** objects in the store. This would help make your reducers simpler and make your features more isolated and reusable.

If you do this, you may find it helpful to divide your actions from a single file into two files—one for actions around posts, and the other around titles.

Other Things To Add

This is a difficult exercise — congratulations if you got here!

There are lots of possible things you could add to this app:

- categories for posts
- multiple blogs, where each blog has many posts *(this would be an ambitious change!)*
- editing comments
- a page listing "most recent comments"
- pagination for the homepage, so it shows batches of titles, rather than all
- authentication and authorization

Solution

View our Solution