Actions

When you call an asynchronous API, there are two crucial moments in time: the moment you start the call, and the moment when you receive an answer (or a timeout).

Each of these two moments usually require a change in the application state; to do that, you need to dispatch normal actions that will be processed by reducers synchronously. Usually, for any API request you'll want to dispatch at least three different kinds of actions:

- **An action informing the reducers that the request began.**

  The reducers may handle this action by toggling an `isFetching` flag in the state. This way the UI knows it's time to show a spinner.

- **An action informing the reducers that the request finished successfully.**

  The reducers may handle this action by merging the new data into the state they manage and resetting `isFetching`. The UI would hide the spinner, and display the fetched data.

- **An action informing the reducers that the request failed.**

  The reducers may handle this action by resetting `isFetching`. Additionally, some reducers may want to store the error message so the UI can display it.

You may use a dedicated `status` field in your actions:

```
{ type: 'FETCH_POSTS' }
{ type: 'FETCH_POSTS', status: 'error', error: 'Oops' }
{ type: 'FETCH_POSTS', status: 'success', response: { ... } }
```

Or you can define separate types for them:

```
{ type: 'FETCH_POSTS_REQUEST' }
{ type: 'FETCH_POSTS_FAILURE', error: 'Oops' }
{ type: 'FETCH_POSTS_SUCCESS', response: { ... } }
```

Choosing whether to use a single action type with flags, or multiple action types, is up to you. It's a convention you need to decide with your team. Multiple types leave less room for a mistake, but this is not an issue if you generate action creators and reducers with a helper library like redux-actions.

Synchronous Action Creators

Let's start by defining the several synchronous action types and action creators we need in our example app. Here, the user can select a subreddit to display:

`actions.js` **(Synchronous)**

```
export const SELECT_SUBREDDIT = 'SELECT_SUBREDDIT'

export function selectSubreddit(subreddit) {
  return {
    type: SELECT_SUBREDDIT,
    subreddit
  }
}
```

They can also press a “refresh” button to update it:

```
export const INVALIDATE_SUBREDDIT = 'INVALIDATE_SUBREDDIT'

export function invalidateSubreddit(subreddit) {
  return {
    type: INVALIDATE_SUBREDDIT,
    subreddit
  }
}
```

These were the actions governed by the user interaction. We will also have another kind of action, governed by the network requests. We will see how to dispatch them later, but for now, we just want to define them.

When it's time to fetch the posts for some subreddit, we will dispatch a `REQUEST_POSTS` action:

```
export const REQUEST_POSTS = 'REQUEST_POSTS'

function requestPosts(subreddit) {
  return {
    type: REQUEST_POSTS,
    subreddit
  }
}
```

It is important for it to be separate from `SELECT_SUBREDDIT` or `INVALIDATE_SUBREDDIT`. While they may occur one after another, as the app grows more complex, you might want to fetch some data independently of the user action (for example, to prefetch the most popular subreddits, or to refresh stale data once in a while). You may also want to fetch in response to a route change, so it's not wise to couple fetching to some particular UI event early on.

Finally, when the network request comes through, we will dispatch `RECEIVE_POSTS`:

```
export const RECEIVE_POSTS = 'RECEIVE_POSTS'

function receivePosts(subreddit, json) {
  return {
    type: RECEIVE_POSTS,
    subreddit,
    posts: json.data.children.map(child => child.data),
    receivedAt: Date.now()
  }
}
```

This is all we need to know for now. The particular mechanism to dispatch these actions together with network requests will be discussed later.

Designing the State Shape

Just like in the basic tutorial, you'll need to design the shape of your application's state before rushing into the implementation. With asynchronous code, there is more state to take care of, so we need to think it through.

This part is often confusing to beginners, because it is not immediately clear what information describes the state of an asynchronous application, and how to organize it in a single tree.

We'll start with the most common use case: lists. Web applications often show lists of things. For example, a list of posts, or a list of friends. You'll need to figure out what sorts of lists your app can show. You want to store them separately in the state, because this way you can cache them and only fetch again if necessary.

Here's what the state shape for our "Reddit headlines" app might look like:

Here's what the state shape for our "Reddit headlines" app might look like:

```
{
  selectedSubreddit: 'frontend',
  postsBySubreddit: {
    frontend: {
      isFetching: true,
      didInvalidate: false,
      items: []
    },
    reactjs: {
      isFetching: false,
      didInvalidate: false,
      lastUpdated: 1439478405547,
      items: [
        {
          id: 42,
          title: 'Confusion about Flux and Relay'
        },
        {
          id: 500,
          title: 'Creating a Simple Application Using React JS and Flux Architect
        }
      ]
    }
  }
}
```

There are a few important bits here:

- We store each subreddit's information separately so we can cache every subreddit. When the user switches between them the second time, the update will be instant, and we won't need to refetch unless we want to. Don't worry about all these items being in memory: unless you're dealing with tens of thousands of items, and your user rarely closes the tab, you won't need any sort of cleanup.

- For every list of items, you'll want to store `isFetching` to show a spinner, `didInvalidate` so you can later toggle it when the data is stale, `lastUpdated` so you know when it was fetched the last time, and the `items` themselves. In a real app, you'll also want to store pagination state like `fetchedPageCount` and `nextPageUrl`.

actionTypes.js × | actions.js | reducers.js | store.js

REACT.js🌈🔥 > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > redux > actionTypes.js > ...

```
1  export const SELECT_SUBREDDIT = "SELECT_SUBREDDIT";
2  export const INVALIDATE_SUBREDDIT = "INVALIDATE_SUBREDDIT";
3  export const FETCH_POSTS_REQUEST = "FETCH_POSTS_REQUEST";
4  export const FETCH_POSTS_SUCCESS = "FETCH_POSTS_SUCCESS";
5  export const FETCH_POSTS_FAILURE = "FETCH_POSTS_FAILURE";
6
```

actionTypes.js | actions.js | reducers.js | store.js

REACT.js 🌈🔥 > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > redux > actions.js > fetchPosts > <function>

```
import {
  INVALIDATE_SUBREDDIT,
  SELECT_SUBREDDIT,
  FETCH_POSTS_REQUEST,
  FETCH_POSTS_SUCCESS,
  //   FETCH_POSTS_FAILURE,
} from "./actionTypes";

//Will fire to set the selectSubreddit..
export const selectSubreddit = (subreddit) => {
  return {
    type: SELECT_SUBREDDIT,
    subreddit,
  };
};

// Will fire to validate the selectSubreddit
export const invalidateSubreddit = (subreddit) => {
  return {
    type: INVALIDATE_SUBREDDIT,
    subreddit,
  };
};
```

```
// will fire to set the loading—> true
export const fetchPostsRequest = (subreddit) => {
  return {
    type: FETCH_POSTS_REQUEST,
    subreddit,
  };
};

//will fire to set the response in the global state
export const fetchPostSuccess = (subreddit, json) => {
  return {
    type: FETCH_POSTS_SUCCESS,
    subreddit,
    posts: json.data.children.map((child) => child.data),
    receivedAt: Date.now(),
  };
};
```

actionTypes.js | actions.js | reducers.js | store.js

REACT.js > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > redux > reducers.js > rootReducer

```
/*
Our Global State Will be see like this..
{
  selectedSubreddit: 'frontend',
  postsBySubreddit: {
    frontend: {
      isFetching: true,
      didInvalidate: false,
      items: []
    },
    reactjs: {
      isFetching: false,
      didInvalidate: false,
      lastUpdated: 1439478405547,
      items: [
        {
          id: 42,
          title: 'Confusion about Flux and Relay'
        },
        {
          id: 500,
          title: 'Creating a Simple Application Using React JS and Flux Architecture'
        }
      ]
    }
  }
}
*/
```

actionTypes.js | actions.js | reducers.js | store.js

REACT.js🌈🔥 > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > redux > reducers.js > ...

```
29  import { combinerReducers } from "redux";
30  import {
31    FETCH_POSTS_REQUEST,
32    FETCH_POSTS_SUCCESS,
33    INVALIDATE_SUBREDDIT,
34    SELECT_SUBREDDIT,
35  } from "./actionTypes";
36
37  //Let's create reducers:
38
39  //$One reducer is for setting the subreddit in the store state;
40  export const selectedSubreddit = (state = "reactjs", action) => {
41    switch (action.type) {
42      case SELECT_SUBREDDIT:
43        return action.selectedSubreddit;
44      default:
45        return state;
46    }
47  };
48
49  //$ Second Reducer for all our API calls updates..and our posts:
50  const initialState = {
51    reactjs: {
52      isFetching: false,
53      didInvalidate: false,
54      items: [],
55    },
56  };
```

actionTypes.js | actions.js | reducers.js | store.js

REACT.js > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > redux > reducers.js

```
export const postsBySubreddit = (state = initialState, action) => {
  switch (action.type) {
    case INVALIDATE_SUBREDDIT:
      return {
        ...state,
        [action.subreddit]: {
          ...state[action.subreddit],
          didInvalidate: true,
        },
      };

    case FETCH_POSTS_REQUEST:
      return {
        ...state,
        [action.subreddit]: {
          ...state[action.subreddit],
          isFetching: true,
          didInvalidate: false,
        },
      };
    case FETCH_POSTS_SUCCESS:
      return {
        ...state,
        [action.subreddit]: {
          ...state[action.subreddit],
          isFetching: false,
          didInvalidate: false,
```

```
            didInvalidate: false,
            items: action.posts,
            lastUpdated: action.receivedAt,
          },
        };
      default:
        return state;
    }
  };

  //Now let's combine reducer:
  const rootReducer = combinerReducers({
    selectedSubreddit,
    postsBySubreddit,
  });

  export default rootReducer;

  //Now next step is to Use async actions creators-->  So let;s get back to our action.js
```

Async Action Creators

Finally, how do we use the synchronous action creators we defined earlier together with network requests? The standard way to do it with Redux is to use the Redux Thunk middleware. It comes in a separate package called `redux-thunk`. We'll explain how middleware works in general later; for now, there is just one important thing you need to know: by using this specific middleware, an action creator can return a function instead of an action object. This way, the action creator becomes a thunk.

When an action creator returns a function, that function will get executed by the Redux Thunk middleware. This function doesn't need to be pure; it is thus allowed to have side effects, including executing asynchronous API calls. The function can also dispatch actions—like those synchronous actions we defined earlier.

We can still define these special thunk action creators inside our `actions.js` file:

actionTypes.js | actions.js | reducers.js | store.js

REACT.js 🌈🔥 > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > redux > actions.js > ...

```
    posts: json.data.children.map((child) => child.data),
    receivedAt: Date.now(),
  };
};

//$ ---------------------- Asynchronous Part ----------------------
// Meet our first thunk action creator!
// Though its insides are different, you would use it just like any other action creator:
// store.dispatch(fetchPosts('reactjs'))
//Using the thunk we gain ability to return function inside our action creator which normally return object..
//And the thunk middleware provide the dispatch method to the returned function as argument, so that inside the function we can dispatch our actions on the basis of certain decision

export const fetchPosts = (subreddit) => {
  // Thunk middleware knows how to handle functions.
  // It passes the dispatch method as an argument to the function,
  // thus making it able to dispatch actions itself.
  return async (dispatch) => {
    // → From here we start our fetching request hence we will call our fetchPostsRequest() action to set the isFetching state to true
    dispatch(fetchPostsRequest(subreddit));

    // now we actually start a fetch call
    const response = await fetch(`https://www.reddit.com/r/${subreddit}.json`);
    const result = response.json;
    return dispatch(fetchUserSuccess(subreddit, result));
  };
};
```

We come here from reducer.js ..

How do we include the Redux Thunk middleware in the dispatch mechanism? We use the `applyMiddleware()` store enhancer from Redux, as shown below:

`index.js`

```
import thunkMiddleware from 'redux-thunk'
import { createLogger } from 'redux-logger'
import { createStore, applyMiddleware } from 'redux'
import { selectSubreddit, fetchPosts } from './actions'
import rootReducer from './reducers'

const loggerMiddleware = createLogger()

const store = createStore(
  rootReducer,
  applyMiddleware(
    thunkMiddleware, // lets us dispatch() functions
    loggerMiddleware // neat middleware that logs actions
  )
)

store.dispatch(selectSubreddit('reactjs'))
store.dispatch(fetchPosts('reactjs')).then(() => console.log(store.getState()))
```

Copy

The nice thing about thunks is that they can dispatch results of each other:

actionTypes.js | actions.js | reducers.js | store.js

REACT.js 🌈🔥 > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > redux > store.js > ...

```
import thunkMiddleware from "redux-thunk";
import { createLogger } from "redux-logger";
import { createStore, applyMiddleware } from "redux";
import { selectSubreddit, fetchPosts } from "./actions";
import rootReducer from "./reducers";

const loggerMiddleware = createLogger();

const store = createStore(
  rootReducer,
  applyMiddleware(
    thunkMiddleware, // lets us dispatch() functions
    loggerMiddleware // neat middleware that logs actions
  )
);

store.dispatch(selectSubreddit("reactjs"));
store.dispatch(fetchPosts("reactjs")).then(() => console.log(store.getState()));

```

Async Flow

Without middleware, Redux store only supports synchronous data flow. This is what you get by default with `createStore()`.

You may enhance `createStore()` with `applyMiddleware()`. It is not required, but it lets you express asynchronous actions in a convenient way.

Asynchronous middleware like redux-thunk or redux-promise wraps the store's `dispatch()` method and allows you to dispatch something other than actions, for example, functions or Promises. Any middleware you use can then intercept anything you dispatch, and in turn, can pass actions to the next middleware in the chain. For example, a Promise middleware can intercept Promises and dispatch a pair of begin/end actions asynchronously in response to each Promise.

When the last middleware in the chain dispatches an action, it has to be a plain object. This is when the synchronous Redux data flow takes place.

Middleware

Redux middleware solves different problems than Express or Koa middleware, but in a conceptually similar way. **It provides a third-party extension point between dispatching an action, and the moment it reaches the reducer.** People use Redux middleware for logging, crash reporting, talking to an asynchronous API, routing, and more.

Understanding Middleware

While middleware can be used for a variety of things, including asynchronous API calls, it's really important that you understand where it comes from. We'll guide you through the thought process leading to middleware, by using logging and crash reporting as examples.

index.js | App.js | SearchBar.js

REACT.js 🌈🔥 > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > index.js

```
1  import React from "react";
2  import ReactDOM from "react-dom";
3  import "./index.css";
4  import App from "./App";
5
6  ReactDOM.render(
7    <React.StrictMode>
8      <App />
9    </React.StrictMode>,
10   document.getElementById("root")
11 );
12
```

index.js | App.js | SearchBar.js

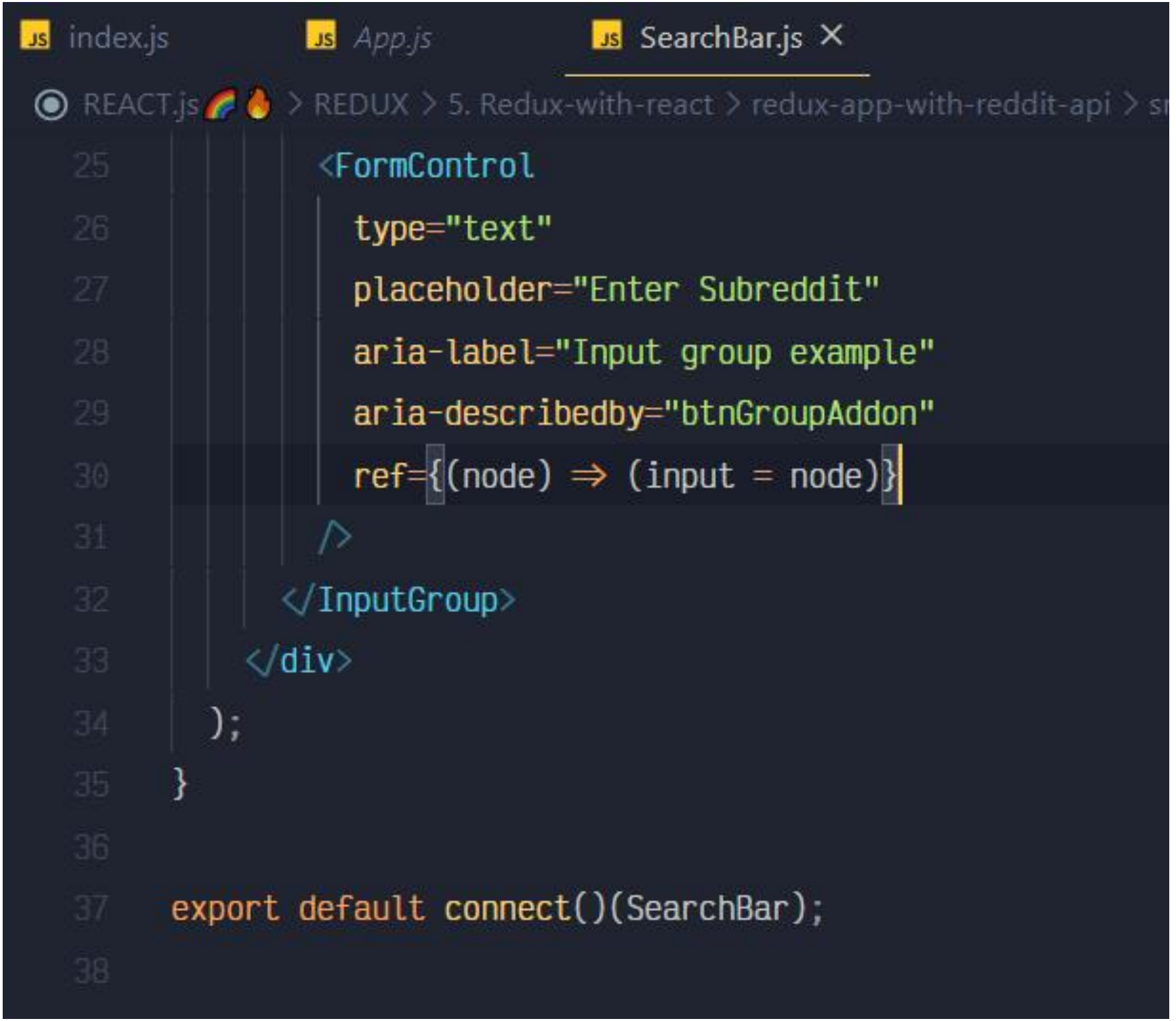
REACT.js🌈🔥 > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > App.js

```
1  import React from "react";
2  import { Container } from "react-bootstrap";
3  import "bootstrap/dist/css/bootstrap.min.css";
4  import SearchBar from "./components/SearchBar";
5  import { Provider } from "react-redux";
6  import store from "./redux/store";
7
8  function App() {
9    return (
10     <Provider store={store}>
11       <Container className="App my-5">
12         <SearchBar />
13       </Container>
14     </Provider>
15   );
16 }
17
18 export default App;
19
```

index.js App.js SearchBar.js

REACT.js🌈🔥 > REDUX > 5. Redux-with-react > redux-app-with-reddit-api > src > components >

```
import React from "react";
import { connect } from "react-redux";
import { InputGroup, FormControl } from "react-bootstrap";
import { BiSearchAlt } from "react-icons/bi";
import { fetchPosts } from "../redux/actions";
let input;
function SearchBar({ dispatch }) {
  const handleClick = (e) => {
    e.preventDefault();
    dispatch(fetchPosts(input.value));
    input.value = "";
  };
  return (
    <div>
      <InputGroup>
        <InputGroup.Prepend>
          <InputGroup.Text
            id="btnGroupAddon"
            onClick={handleClick}
            style={{ cursor: "pointer" }}
          >
            <BiSearchAlt />
          </InputGroup.Text>
        </InputGroup.Prepend>
```


index.js
App.js
SearchBar.js
REACT.js > REDUX > 5. Redux-with-react > redux-app-with-reddit-api
<FormControl
type="text"
placeholder="Enter Subreddit"
aria-label="Input group example"
aria-describedby="btnGroupAddon"
ref={(node) => (input = node)}
/>
</InputGroup>

);
}
export default connect()(SearchBar);