Redux

Actions are payloads of information that send data from your application to your store.

They are the only source of information for the store.

You send them to the store using store.dispatch().

Actions are plain Javascript object.

Actions must have a type property that indicates the type of action being performed.

Types should typically be defined as string constants.

Reducers specify how the application’s state change in response to actions sent to the store.

Remember that actions only describe what happened, but don’t describe hw the application’s state changes

In redux, all the application state it stored as a single object.

What’s the minimal representation of your app’s state as an object.

The reducer is a pure function that takes the previous state and an action, and returns the next state

In the previous sections, we defined the actions that represent the fact about “what happened” and the reducers that update the state according to those actions

The store is the object that brings them together. The store has the following responsibilities:

* Holds application state
* Allows access to state via getState()
* Allows state to be updated via dispatch(action)
* Registers listeners via subscribe(listener)
* Handles unregistering of listeners via the function returned by

It’s important to note that you’ll only have a single store in redux application.

When you want to split your data handling logic, you’ll use reducer composition instead of many stores.

It’s easy to create a store if you have a reducer.

We use combineReducers() to combine several reducers into one and pass it to createStore()

A container component is just a react component that use store.subcribe() to read a part of the redux state tree and supply props to a presentational component it renders

You could write a container component by hand, but we suggest instead generating container components with react redux library’s connect() function.

To use connect(), you need to define a special function called mapStateToProps() that describes how to transform the current redux store state into the props you want to pass to a presentational component you are wrapping

You can define a function called mapDispatchToProps() that receives the dispatch() method and return callback props that you want to inject into the presentational component.