Let us assume our application’s state is described by a plain object called **initialState** which is as follows −

const initialState = {

isLoading: false,

items: [],

hasError: false

};

Every piece of code in your application cannot change this state. To change the state, you need to dispatch an action.

What is an action?

An action is a plain object that describes the intention to cause change with a type property. It must have a type property which tells what type of action is being performed. The command for action is as follows −

return {

type: 'ITEMS\_REQUEST', //action type

isLoading: true //payload information

}

Actions and states are held together by a function called Reducer. An action is dispatched with an intention to cause change. This change is performed by the reducer. Reducer is the only way to change states in Redux, making it more predictable, centralised and debuggable. A reducer function that handles the ‘ITEMS\_REQUEST’ action is as follows −

const reducer = (state = initialState, action) => { //es6 arrow function

switch (action.type) {

case 'ITEMS\_REQUEST':

return Object.assign({}, state, {

isLoading: action.isLoading

})

default:

return state;

}

}

Redux has a single store which holds the application state. If you want to split your code on the basis of data handling logic, you should start splitting your reducers instead of stores in Redux.

We will discuss how we can split reducers and combine it with store later in this tutorial.

Redux components are as follows −

