The need of actions

Redux reducers are without doubt the most important concept in Redux. Reducers produce the state of an application.

But how does a reducer know when to generate the next state?

The second principle of Redux says the only way to change the state is by sending a signal to the store. This signal is an action. So "**dispatching an action**" means **sending out a signal** to the store. The reassuring thing is that Redux actions are nothing more than JavaScript objects. This is how an action looks like:

```
type: "INCREMENT_COUNTER",
 payload: {
   value: 1
 }
```

The *type* property drives how the state should change and it's always required by Redux. The *payload* property instead describes what should change, and might be omitted if you don't have new data to save in the store.

Create actions:

As a best practice in Redux we wrap every action within a function, so that object creation is abstracted away. Such function takes the name of action creator: let's put everything together by creating a simple action creator.

In the directory Actions, create a file named actions.js

```
// src/Actions/actions.js


export const incrementCounter = (payload) => ({
 type: "INCREMENT_COUNTER",
```

```
  payload
});
```

Create actions

You can notice that the type property is a string. Strings are prone to typos and duplicates and for this reason it's better to declare actions as constants. Here come the role of the directory constants. Under this folder create a new file `actions-types.js`

```
// src/js/constants/action-types.js
export const INCREMENT_COUNTER = "INCREMENT_COUNTER";
```

Now open up again src/js/Actions/actions.js and update the action to use action types:

```
// src/Actions/actions.js

import { INCREMENT_COUNTER } from "../constants/actions-types";

export const incrementCounter = (payload) => ({
 type: INCREMENT_COUNTER,
 payload
}
```

Refactoring the reducer

When an action is dispatched, the store forwards a message (the action object) to the reducer. At this point, the reducer checks the type of this action. Then depending on the action type, the reducer produces the next state eventually merging the action payload into the new state.

Let's fix our `rootReducer.js`

```
import { INCREMENT_COUNTER } from "../constants/actions-types";

const initialState = {
  counter: 0
};
```

```
function rootReducer(state = initialState, action) {
switch (action.type) {
  case INCREMENT_COUNTER:


  return {
    counter : state.counter + 1
  }
  return state;
};

export default rootReducer;
```

Previous next