0x08. React Redux reducer+selector

Front-end JavaScript ES6 React

Weight: 1

Project over - took place from Oct 3, 2024 6:00 AM to Oct 10, 2024 6:00 AM

Manual QA review was done on Oct 12, 2024 8:42 PM

In a nutshell…

- **Manual QA review:** 45.0/45 mandatory
- **Altogether: 100.0%**
  - Mandatory: 100.0%
  - Optional: no optional tasks

Overall comment:
Great work done. Keep up with the pace.

Resources

Read or watch:

- Reducers
- Selectors
- Writing tests
- Immutable Map documentation
- Normalizr
- Normalizing State Shape

Learning Objectives

At the end of this project, you should be able to explain to anyone, **without the help of Google**:

- The purpose of a reducer and the role it plays within your application
- Why a reducer should stay as pure as possible
- Why mutations should not happen within a reducer
- The use of Immutable within the reducer
- The use of Normalizr within the app
- Selectors: what they are and when to use them

Requirements

- Allowed editors: `vi`, `vim`, `emacs`, `Visual Studio Code`
- All your files should end with a new line

- All your files will be interpreted/compiled on Ubuntu 18.04 LTS using node `12.x.x` and `npm 6.x.x`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Push all of your files, including `package.json` and `.babelrc`
- All of your functions must be exported

Tasks

0. Write a basic reducer

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

- Reuse the latest dashboard project you worked on in the React course `0x08-React_Redux_action_creator+normalizr`

Create the basic state

In a file named `reducers/uiReducer.js`, define the initial state of the reducer for the UI:

- It should have one boolean property `isNotificationDrawerVisible`
- It should have one boolean property `isUserLoggedIn`
- It should have one empty object `user`

Create the reducer function

In the same file, import all the actions that you created in the file `actions/uiActionTypes` and create a reducer function named `uiReducer`:

- `DISPLAY_NOTIFICATION_DRAWER` should set `isNotificationDrawerVisible` to `true`
- `HIDE_NOTIFICATION_DRAWER` should set `isNotificationDrawerVisible` to `false`
- `LOGIN_SUCCESS` should set `isUserLoggedIn` to `true`
- `LOGIN_FAILURE` should set `isUserLoggedIn` to `false`
- `LOGOUT` should set `isUserLoggedIn` to `false`

Write the tests

In a file named `reducers/uiReducer.test.js`, define the test suite for our simple reducer:

- Write a test verifying the state returned by the `uiReducer` function equals the initial state when no action is passed
- Write a test verifying the state returned by the `uiReducer` function equals the initial state when the action `SELECT_COURSE` is passed
- Write a test verifying the state returned by the `uiReducer` function, when the action `DISPLAY_NOTIFICATION_DRAWER` is passed, changes correctly the `isNotificationDrawerVisible` property

Tips:

- Don't forget to set up the default case in your switch function

Requirements:

- You should not mutate the state within the reducer
- You must use the spread operator to change the state
- All the tests in the project should pass

Repo:

- GitHub repository: `alx-react`
- Directory: `0x08-react_redux_reducer_selector`
- File: `task_0/dashboard/src/reducers/uiReducer.js, task_0/dashboard/src/reducers/uiReducer.test.js`

View results

1. Use Immutable for the UI Reducer

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Now that you have set up a basic reducer, let's reuse what we learned in the Immutable module and apply it to that reducer:

- Install `Immutable.js` within the project
- Update the `uiReducer.js` file to use `Map` from `Immutable.js`
- Update the different part of the reducer function to use `set` from `Map`
- Update the test suite, so it takes into account the changes

Tips:

- You can use the `toJS` function within your tests for an easy comparison
- Remember that `Immutable.js` always return a new Map after a modification

Requirements:

- For better performances, do not use any `fromJS`, `toJS` function within the reducer
- All the tests in the project should pass

Repo:

- GitHub repository: `alx-react`
- Directory: `0x08-react_redux_reducer_selector`
- File: `task_1/dashboard/src/reducers/uiReducer.js, task_1/dashboard/src/reducers/uiReducer.test.js`

View results

2. Create a reducer for Courses

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Create a load action

In the `courseActionTypes` file, create a new action corresponding to when the API returns the list of courses. You can name it `FETCH_COURSE_SUCCESS`

Create the course reducer and default state

In a file `courseReducer.js`, write a reducer function. The default state should be an empty array.

Define the `FETCH_COURSE_SUCCESS` action

When the action creator sends the action `FETCH_COURSE_SUCCESS`, it also sends the list of courses in a data attribute. The action would look like:

```
{
  type: FETCH_COURSE_SUCCESS,
  data: [
    {
      id: 1,
      name: "ES6",
      credit: 60
    },
    {
      id: 2,
      name: "Webpack",
      credit: 20
    },
    {
      id: 3,
      name: "React",
      credit: 40
    }
  ]
}
```

When updating the state of the reducer, you should also set the attribute `isSelected` to false for every item in the list. The expected data from the reducer should be:

```
[
  {
    id: 1,
    name: "ES6",
    isSelected: false,
    credit: 60
  },
  {
    id: 2,
    name: "Webpack",
    isSelected: false,
    credit: 20
  },
  {
    id: 3,
    name: "React",
    isSelected: false,
    credit: 40
  }
]
```

Define the `SELECT_COURSE` and `UNSELECT_COURSE` actions

When the action creator sends the action `SELECT_COURSE`, it also sends an index corresponding to the id of the course to update. The action would look like:

```
{
  type: SELECT_COURSE,
  index: 2
}
```

The expected data from the reducer should be:

```
[
  {
    id: 1,
    name: "ES6",
    isSelected: false,
    credit: 60
```

```
  },
  {
    id: 2,
    name: "Webpack",
    isSelected: true,
    credit: 20
  },
  {
    id: 3,
    name: "React",
    isSelected: false,
    credit: 40
  }
]
```

When the action creator sends the action `UNSELECT_COURSE`, it also sends an index corresponding to the id of the course to update. The action would look like:

```
{
  type: UNSELECT_COURSE,
  index: 2
}
```

The expected data from the reducer should be:

```
[
  {
    id: 1,
    name: "ES6",
    isSelected: false,
    credit: 60
  },
  {
    id: 2,
    name: "Webpack",
    isSelected: false,
    credit: 20
  },
  {
    id: 3,
    name: "React",
    isSelected: false,
    credit: 40
  }
]
```

Write the tests

In a `courseReducer.test.js`, write a test suite for the new reducer. Define the following tests:

- Test that the default state returns an empty array
- Test that `FETCH_COURSE_SUCCESS` returns the data passed
- Test that `SELECT_COURSE` returns the data with the right item updated
- Test that `UNSELECT_COURSE` returns the data with the right item updated

Tips:

- Use ES6 for this reducer, we can look at Immutable later

Requirements:

- Try to make the update of object as efficient as possible, for example you can use ES6 Map
- All the tests in the project should pass

Repo:

- GitHub repository: `alx-react`
- Directory: `0x08-react_redux_reducer_selector`
- File: `task_2/dashboard/src/actions/courseActionTypes.js, task_2/dashboard/src/reducers/courseReducer.js, task_2/dashboard/src/reducers/courseReducer.test.js`

View results

3. Create the reducer for notifications

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Create a load action

In the `notificationActionTypes` file, create a new action corresponding to when the API returns the list of notifications. You can name it `FETCH_NOTIFICATIONS_SUCCESS`

Create the notifications reducer and default state

In a file `notificationReducer.js`, write a reducer function. The default state should be an object with:

- `notifications`, which will store the list of notifications
- `filter`, which will be the attribute storing which filter is selected

Define the `FETCH_NOTIFICATIONS_SUCCESS` action

When the action creator sends the action `FETCH_NOTIFICATIONS_SUCCESS`, it also sends the list of notifications in a data attribute. The action would look like:

```
{
  type: FETCH_NOTIFICATIONS_SUCCESS,
  data: [
    {
      id: 1,
      type: "default",
      value: "New course available"
    },
    {
      id: 2,
      type: "urgent",
      value: "New resume available"
    },
    {
      id: 3,
      type: "urgent",
      value: "New data available"
    }
  ]
}
```

When updating the state of the reducer, you should also set the attribute `isRead` to false for every item in the list. The expected data from the reducer should be:

```
{
  filter: "DEFAULT",
  notifications: [
    {
      id: 1,
      isRead: false,
      type: "default",
      value: "New course available"
    },
    {
      id: 2,
      isRead: false,
      type: "urgent",
      value: "New resume available"
    },
    {
      id: 3,
      isRead: false,
      type: "urgent",
      value: "New data available"
    }
  ]
}
```

Define the `MARK_AS_READ` action

When the action creator sends the action `MARK_AS_READ`, it also sends an index corresponding to the id of the notification to update. The action would look like:

```
{
  type: MARK_AS_READ,
  index: 2
}
```

The expected data from the reducer should be:

```
{
  filter: "DEFAULT",
  notifications: [
    {
      id: 1,
      isRead: false,
      type: "default",
      value: "New course available"
    },
    {
      id: 2,
      isRead: true,
      type: "urgent",
      value: "New resume available"
    },
    {
      id: 3,
      isRead: false,
      type: "urgent",
      value: "New data available"
    }
```

```
  ]
}
```

Define the `SET_TYPE_FILTER` action

When the action creator sends the action `SET_TYPE_FILTER`, it also sends a filter attribute with either `DEFAULT` or `URGENT`. The action would look like:

```
{
  type: SET_TYPE_FILTER,
  filter: "URGENT"
}
```

The expected data from the reducer should be:

```
{
  filter: "URGENT",
  notifications: [
    {
      id: 1,
      isRead: false,
      type: "default",
      value: "New course available"
    },
    {
      id: 2,
      isRead: false,
      type: "urgent",
      value: "New resume available"
    },
    {
      id: 3,
      isRead: false,
      type: "urgent",
      value: "New data available"
    }
  ]
}
```

Tips:

- Use ES6 for this reducer, we can look at Immutable later

Requirements:

- Try to make the update of object as efficient as possible, for example you can use ES6 Map
- All the tests in the project should pass

Repo:

- GitHub repository: `alx-react`
- Directory: `0x08-react_redux_reducer_selector`
- File: `task_3/dashboard/src/actions/notificationActionTypes.js, task_3/dashboard/src/reducers/notificationReducer.js, task_3/dashboard/src/reducers/notificationReducer.test.js`

View results

4. Normalizr & Immutable

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

As you can see, updating a specific item in a collection is rather complicated and error prone. Using Normalizr is a good opportunity to simplify mutation

Course schema

Create a new file `schema/courses.js`. In the file define a schema entity for `courses`. Create a function `coursesNormalizer` that would take `data` as argument and normalize the data with the schema you created.

In the course reducer function:

- Update the initial state to use an `Immutable.js` Map
- When `FETCH_COURSE_SUCCESS` action is called, normalize the data with the function you created and merge it with the state
- When `SELECT_COURSE` or `UNSELECT_COURSE` is called, use the `setIn` function from Immutable to update the value of the item

Update the notification schema

In the file `schema/notifications.js`, create a function `notificationsNormalizer` that would take `data` as argument and normalize it with the notification schema you created in the previous course.

Update the notification reducer

In the notification reducer function:

- Update the initial state to use an `Immutable.js` Map
- When `FETCH_NOTIFICATIONS_SUCCESS` action is called, normalize the data with the function `notificationsNormalizer` you created and merge it with the state
- When `SET_TYPE_FILTER`, use the `set` function from Immutable to update the value of the state
- When `MARK_AS_READ`, use the `setIn` function from Immutable to update the value of the item in the state

Update the test files/suites:

- Update the course reducer test file to match the new reducer

Tips:

- You can use the `fromJS` function from `Immutable.js` to easily create the initial state from an object
- You can use the `toJS` function from `Immutable.js` to easily compare the expected data
- Selecting an unselecting a course item should only take one line now
- Marking a notification item as read should only take one line now

Requirements:

- All the tests in the project should pass

Repo:

- GitHub repository: `alx-react`
- Directory: `0x08-react_redux_reducer_selector`
- File: `task_4/dashboard/src/schema/courses.js, task_4/dashboard/src/reducers/courseReducer.js, task_4/dashboard/src/schema/notifications.js, task_4/dashboard/src/reducers/notificationReducer.js, task_4/dashboard/src/reducers/courseReducer.test.js, task_4/dashboard/src/reducers/notificationReducer.test.js`

View results

5. Selectors

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Selectors are an efficient way to access the data from the state because a selector is not recomputed unless one of its arguments change.

Let's create a few selectors for the Notifications reducer in `src/selectors/notificationSelector.js`

- Create a first selector for the filter named `filterTypeSelected`, that will return the value of the filter
- Create another selector for the notifications named `getNotifications`, that will return the list of notifications in a Map format
- Create another selector for the notifications named `getUnreadNotifications`, that will return the list of unread notifications in a Map format

Create a test suite for your selectors in a file named `src/selectors/notificationSelector.test.js`:

- test that `filterTypeSelected` works as expected
- test that `getNotifications` returns a list of the message entities within the reducer
- test that `getUnreadNotifications` return a list of the message entities within the reducer

Tips:

- To write your tests, you can have a state variable using the reducer you created. And pass the state to the selector functions
- You can also look into using Reselect for your own projects when you have advanced needs for filtering, reducing and merging data from the state

Requirements:

- All the tests in the project should pass

Repo:

- GitHub repository: `alx-react`
- Directory: `0x08-react_redux_reducer_selector`
- File: `task_5/dashboard/src/selectors/notificationSelector.js, task_5/dashboard/src/selectors/notificationSelector.test.js`

View results

Ready for a new review