Title: SIMPLE REDUX CREATE DELETE CONTACT APPLICATION

Answer:

SIMPLE REDUX CREATE DELETE CONTACT APPLICATION

Simple Redux Create Delete Contact Application is today's small project. We will use **ReactJS** as a frontend. We are not using any backend in this tutorial, so a just client-side application. Redux is the most famous library to manage the state at the client side, and it is trendy among React community. We will build a Simple Contact Create Delete Application. In that user can add a new contact and it will display as a list and user can also delete any contact.

First, we need to download the create-react-app globally on the PC and then we can begin our project.

STEP 1: DO ONE REACTJS PROJECT.

Type the following command one by one.

```
npm install -g create-react-app
create-react-app redux-contact-app
```

It will make some boilerplate for our application and also install all the dependencies for our project. Still, we need to establish some other dependencies as well.

```
npm install --save redux react-redux
```

So, this installs redux and react-redux dependencies.

STEP 2: WE NEED TO CONFIGURE THE REDUX STORE.

First, we need to make three folders inside **src** directory.

1. actions
2. reducers
3. store

To set the store, we need to create one file inside store folder called **configureStore.js**.

```
// configureStore.js

import { createStore } from 'redux';

export default function configureStore(initialState) {
    return createStore();
}
```

Also, we need to wrap the Provider component Outside our **App.js** component in the **index.js** file.

```
// index.js

import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { Provider } from 'react-redux';

import registerServiceWorker from './registerServiceWorker';

ReactDOM.render(
<Provider>
    <App />
</Provider>, document.getElementById('root'));
registerServiceWorker();
```

The react-redux library provides us Provider, which wraps the App component and make the available store to the context of our application.

STEP 3: WE NEED TO MAKE ONE FORM TO ADD THE CONTACT NAME.

Go to the **App.js** file and put the following code in it.

```
// App.js

import React, { Component } from 'react';

class App extends Component {

  constructor(props){
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.state = {
      name: ''
    }
  }

  handleChange(e){
    this.setState({
      name: e.target.value
    })
  }

  handleSubmit(e){
    e.preventDefault();
    console.log(this.state.name);
  }

  render() {
    let name;
    return(
      <div>
        <h1>Clientside Contacts Application</h1>
        <hr />
        {/* <ul>
          {this.props.contacts.map((contact, i) => <li key={i}>{contact.nam
e}</li> )}
        </ul> */}
        <div>
          <h3>Add Contact Form</h3>
          <form onSubmit={this.handleSubmit}>
            <input type="text" onChange={this.handleChange} />
            <input type="submit" />
          </form>
        </div>
      </div>
    )
  }
}

export default App;
```

So, it will create a form and when we submit the data, the data logs in the console. **Now, if we need to change the add the new contact in the array of contacts or modify it, we do not do it directly. We need to dispatch the actions and that actions call the reducer and reducer returns a new state of the application.**

STEP 4: MAKE ACTIONTYPE.JS CONSTANTS FILE.

We need to make one file called actionType.js inside the actions folder.

```
// actionTypes.js

export const GET_ALL_CONTACTS = 'GET_ALL_CONTACTS';
export const CREATE_NEW_CONTACT = 'CREATE_NEW_CONTACT';
```

This file exports our actionType for every action object. So every time, if we want to dispatch any actions then we access the actions quickly to avoid any typos in action types.

STEP 5: MAKE ONE ACTION OBJECT FOR CREATING CONTACT.

Inside actions folder, make one file called **contactAction.js**.

```
// contactAction.js

import * as actionTypes from './actionTypes';

export const createContact = (contact) => {
    return {
      type: actionTypes.CREATE_NEW_CONTACT,
      contact: contact
    }
  };
```

createContact() function returns an object that describes two things.

1. action type
2. payload

STEP 6: MAKE ONE REDUCER INSIDE REDUCERS FOLDER.

Reducers are used to update the state object in your store. Just like actions, your application can have multiple reducers.

```
// contactReducer.js

import * as actionTypes from '../actions/actionTypes';

export default (state = [], action) => {
    switch (action.type){

      case actionTypes.CREATE_NEW_CONTACT:
      return [
        ...state,
        Object.assign({}, action.contact)
      ];
      default:
            return state;
    }
  };
```

Here, one thing to note that, we have not mutated the state directly instead we have returned a new state.

So, the new state is original state array + new contact data and returns a new collection.

STEP 7: CREATE COMBINEREDUCER FUNCTION.

Inside **reducers** folder, make one file called **index.js** file.

```
// index.js

import { combineReducers } from 'redux';
import contacts from './contactReducer';

export default combineReducers({
    contacts: contacts
});
```

Now, update the **configureStore** function and pass the **rootReducer** to it.

```
// configureStore.js

import {createStore} from 'redux';
import rootReducer from '../reducers';

export default function configureStore(initialState) {
  return createStore(rootReducer, initialState);
}
```

Also, we need to pass the store in our application. So we need also to update the **index.js** file.

```
// index.js

import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { Provider } from 'react-redux';
import registerServiceWorker from './registerServiceWorker';
import configureStore from './store/configureStore';

const store = configureStore();

ReactDOM.render(
<Provider store={store}>
    <App />
</Provider>, document.getElementById('root'));
registerServiceWorker();
```

STEP 8: WE NEED TO CONNECT THIS STORE TO THE APP.JS COMPONENT.

If we want to access all the contacts data from the store, then we must the store state as props to the react components.

Connect() function bridge the gap between store and components and provide a way to pass state as props to display data or dispatch any actions to the Redux store.

```
// App.js

import React, { Component } from 'react';
import { connect } from 'react-redux';
import * as contactAction from './actions/contactAction';

class App extends Component {

  constructor(props){
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.state = {
      name: ''
    }
  }

  handleChange(e){
    this.setState({
      name: e.target.value
    })
  }

  handleSubmit(e){
    e.preventDefault();
    let contact = {
      name: this.state.name
    }
    this.props.createContact(contact);
  }

  render() {

    return(
      <div>
        <h1>Clientside Contacts Application</h1>
        <hr />
        { <ul>
          {this.props.contacts.map((contact, i) => <li key={i}>{contact.nam
e}</li> )}
        </ul> }
        <div>
          <h3>Add Contact Form</h3>
          <form onSubmit={this.handleSubmit}>
            <input type="text" onChange={this.handleChange} />
            <input type="submit" />
          </form>
        </div>
      </div>
    )
  }
}

const mapStateToProps = (state, ownProps) => {
  return {
    contacts: state.contacts
  }
};

const mapDispatchToProps = (dispatch) => {
  return {
    createContact: contact => dispatch(contactAction.createContact(contact))
  }
};

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

Now, we can add the contacts, and it will list down the contact one by one from the store.

STEP 9: MAKE AN ACTION TO REMOVE THE CONTACT LIST AND STYLE THE PROJECT.

First, update the **actionType.js** file and add new action type.

```
// actionType.js

export const GET_ALL_CONTACTS = 'GET_ALL_CONTACTS';
export const CREATE_NEW_CONTACT = 'CREATE_NEW_CONTACT';
export const REMOVE_CONTACT = 'REMOVE_CONTACT';
```

We need to make an action described object in the **contactAction.js** file.

```
// contactAction.js

import * as actionTypes from './actionTypes';

export const createContact = (contact) => {
    return {
      type: actionTypes.CREATE_NEW_CONTACT,
      contact: contact
    }
  };

export const deleteContact = (id) => {
    return {
        type: actionTypes.REMOVE_CONTACT,
        id: id
    }
}
```

Now, add a new case in the **contactReducer.js** file to delete the contact.

```
// contactReducer.js

import * as actionTypes from '../actions/actionTypes';

export default (state = [], action) => {
    switch (action.type){
      case actionTypes.CREATE_NEW_CONTACT:
      return [
        ...state,
        Object.assign({}, action.contact)
      ];
      case actionTypes.REMOVE_CONTACT:
      return state.filter((data, i) => i !== action.id);
      default:
            return state;
    }
  };
```

In here, I have used the **ES6 Filter** function to remove the item list match with the id we just pass as a payload.

Finally, our **App.js** file looks like this with bootstrap styles.

```
// App.js

import React, { Component } from 'react';
import { connect } from 'react-redux';
import * as contactAction from './actions/contactAction';

class App extends Component {

  constructor(props){
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);

    this.state = {
      name: ''
    }
  }

  handleChange(e){
    this.setState({
      name: e.target.value
    })
  }

  handleSubmit(e){
    e.preventDefault();
    let contact = {
      name: this.state.name
    }
    this.setState({
      name: ''
    });
    this.props.createContact(contact);
  }

  listView(data, index){
    return (
      <div className="row">
        <div className="col-md-10">
          <li key={index} className="list-group-item clearfix">
            {data.name}
          </li>
        </div>
        <div className="col-md-2">
          <button onClick={(e) => this.deleteContact(e, index)} className="b
tn btn-danger">
            Remove
          </button>
        </div>
    </div>
    )
  }

  deleteContact(e, index){
    e.preventDefault();
    this.props.deleteContact(index);
  }

  render() {

    return(
      <div className="container">
        <h1>Clientside Contacts Application</h1>
        <hr />
        <div>
          <h3>Add Contact Form</h3>
          <form onSubmit={this.handleSubmit}>
            <input type="text" onChange={this.handleChange} className="form-
control" value={this.state.name}/><br />
            <input type="submit" className="btn btn-success" value="ADD"/>
          </form>
          <hr />
        { <ul className="list-group">
          {this.props.contacts.map((contact, i) => this.listView(contact,
i))}
        </ul> }
        </div>
      </div>
    )
  }
}

const mapStateToProps = (state, ownProps) => {
  return {
    contacts: state.contacts
  }
};

const mapDispatchToProps = (dispatch) => {
  return {
    createContact: contact => dispatch(contactAction.createContact(contac
t)),
    deleteContact: index =>dispatch(contactAction.deleteContact(index))
  }
};

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

Tags: react