REACT

REDUX

Introduction to Redux

Redux is a popular JavaScript framework that provides a predictable state container for applications. Redux is based on a simplified version of Flux, a framework developed by Facebook. Unlike standard MVC frameworks, where data can flow between UI components and storage in both directions, Redux strictly allows data to flow in one direction only.

In Redux, all data – i.e. state – is held in a container known as the store. There can only be one of these within an application. The store is essentially a state tree where states for all objects are kept. Any UI component can access the state of a particular object directly from the store. To change a state from a local or remote component, an action needs to be dispatched. Dispatch in this context means sending actionable information to the store. When a store receives an action, it delegates it to the relevant reducer. A reducer is simply a pure function that looks at the previous state, performs an action and returns a new state.

Setting Up Redux

To install redux, run

$ npm install redux

Create Store

Let's start by creating a student state and later put it in a store. To achieve this, we will follow these steps:

1. create a Students file
2. In the Student file, import {createStore} from 'redux'
3. Create a state

```
const studentState = {
  students:[
    {'id':1, 'name':'Opeyemi', 'score':10, 'nationality':'Nigeria'},
    {'id':2, 'name':'Oyeronke', 'score':15, 'nationality':'Ghana'},
    {'id':3, 'name':'Akanbi', 'score':12, 'nationality':'Namibia'}
  ],
}
```

4. Create a reducer

```
const studentReducer = (state=studentState, action)=>{
  return state;
}
```

the action contains a logic to perform CRUD and a payload(data for updating state)

5. Pass the reducer to the createState() function

const store = createStore(studentReducer);

6. Finally, you export studentStore

export default store

Then, in the index.js file, we have

import ReactDOM from 'react-dom';
import store from './students'

console.log(store.getState());
const name = 'Redux'
ReactDOM.render(name, document.getElementById('root'));
Then, check the console of your browser to view the result.

Combine Reducers

If you have more than one reducer, you would need to use combineReducers to bind the reducers together before creating a store.
Let's explore a simple example of this:
1. Create a CombineReducers component
2. In the CombineReducers component,
import {createStore, combineReducers} from 'react-redux'
3. Create two states: studentState and TeacherState

```
//create an initial state
const studentState = {
  students:[
    {'id':1, 'name':'Opeyemi', 'score':10, 'nationality':'Nigeria'},
    {'id':2, 'name':'Oyetunde', 'score':15, 'nationality':'Ghana'},
    {'id':3, 'name':'Oyekunle', 'score':12, 'nationality':'Namibia'}
  ],
}

//create an initial state
const teacherState = {
  teachers:[
    {'id':1, 'name':'Odunola', 'age':14, 'nationality':'England'},
    {'id':2, 'name':'Damilola', 'age':35, 'nationality':'Germany'},
    {'id':3, 'name':'Arike', 'age':22, 'nationality':'Egypt'}
  ]
}
```

4. Create two reducers: studentReducer and teacherReducer

```
const studentReducer = (state=studentState, action)=>{
  return state;
}
const teacherReducer = (state=teacherState, action)=>{
  return state;
}
```

5. Store the two reducers in an object:

```
//the studentReducer and teacherReducer are assigned to collegeReducers
object
const collegeReducers = {
  studentReducer,
  teacherReducer
}
```

6. Combine the reducers or prepare the reducers for createStore() function
const reducers = combineReducers(collegeReducers)
7. Pass the reducer to the createState() function
const store = createStore(reducers);
8. Finally, you export the store
export default store
Then, in the index.js file, we have
import ReactDOM from 'react-dom';
import store from './combineReducers'

console.log(store.getState);
const name = 'Redux'
ReactDOM.render(name, document.getElementById('root'));

Adding Payload to State

To add payload to a state, we would need to create an action and dispatch the action. Let's update the Students file as thus:

1. After the studentState, create an action(function) that returns the action type and a payload

```
//create action
const addStudent = (id, name, score, nationality)=>{
  return {
    type: 'ADD_STUDENT',
    record:{id, name, score, nationality}
  }
}
```

2. Update the studentState like this:

```
// create a student reducer function with parameters state and action
//the action contains a logic to perform CRUD and a payload(data for updating
state)
const studentReducer = (state=studentState, action)=>{
  if (action.type === 'ADD_STUDENT') {
      return{
        ...state,
        students:[...state.students, action.record]
      }
    }
  return state;
}
```

3. Pass action to the dispatch function.

```
store.dispatch(addStudent(4, 'Wale', 19, 'Cameroon'));
```

4. export default store

Removing Payload from State

This is also similar to our previous example, the only difference is the type of action needed.

Let's play with this example.

1. After the addStudent, create an action(function) that returns the action type and a payload

```
//create a delete action
const deleteStudent = (id)=>{
  return{
    type: 'DELETE_STUDENT',
    record:{id}
  }
}
```

2. Update the studentState like this:

```
// create a student reducer function with parameters state and action
//the action contains a logic to perform CRUD and a payload(data for updating
state)
const studentReducer = (state=studentState, action)=>{
  if (action.type === 'ADD_STUDENT') {
      return{
        ...state,
        students:[...state.students, action.record]
      }
    }
    if (action.type === 'DELETE_STUDENT') {
      return{
        ...state,
        students:state.students.filter(student=>student.id !==
action.record.id)
      }
    }
  return state;
}
```

3. Pass deleteStudent action to the dispatch function.

store.dispatch(deleteStudent(1));

4. export default store

Updating the State

1. After the addStudent, create an action(function) that returns the action type and a payload

```
//create an update action
const updateStudent = (id, name, score, nationality)=>{
  return {
    type: 'UPDATE_STUDENT',
    record:{id, name, score, nationality}
  }
}
```

2. Update the studentState like this:

```
// create a student reducer function with parameters state and action
//the action contains a logic to perform CRUD and a payload(data for updating
state)
const studentReducer = (state=studentState, action)=>{
  if (action.type === 'ADD_STUDENT') {
      return{
        ...state,
        students:[...state.students, action.record]
      }
    }
    if (action.type === 'DELETE_STUDENT') {
      return{
        ...state,
        students:state.students.filter(student=>student.id !==
action.record.id)
      }
    }
if (action.type === 'UPDATE_STUDENT') {
      return{
        ...state,
        students:state.students.map(student=>{
          if (student.id === action.record.id) {
            return action.record;
          }else {
            return student;
          }
        })
      }
    }
  return state;
}
```

3. Pass updateStudentaction to the dispatch function.

store.dispatch(updateStudent(2, 'Wale', 19, 'Gabon'));

4. export default store

Map State to Props

How do we combine Redux and React together? Don't be scared, it's very simple. This simple example will prove that. Let's explore a simple example of how we can map the state initiated in Redux and pass it as props to React components.

```
Let's update our index.js file like this
import React from 'react';
import ReactDOM from 'react-dom';
import {Provider} from 'react-redux'
import store from './Students'
import StudentListfrom './StudentList'

//setting up redux in react
ReactDOM.render(<Provider store={store}><StudentList/></Provider>,
document.getElementById('root'));
```

The Provider allows us to pass store to React.

Next,

1. Create a StudentList component

2. Add this to the StudentList file

```
import React, {Component} from 'react'
class StudentList extends Component{
  render(){
    return(
      <div>
        <h2>List of students</h2>
        <table>
          <tbody>
          </tbody>
        </table>
      </div>
    )
  }
}
export default StudentList
```

3. How do we now pass props to the StudentList component? We are going to make use of Redux's built-in HOC

We need to import connect

import {connect} from 'react-redux'

Then, map the state to studentList component, add this lines of codes to replace export default StudentList

```
const mapStateToProps = (state)=>{
  return{
    students: state.studentReducer.students
  }
}
export default connect(mapStateToProps)(StudentList );
```

After super-charging the StudentList component, the students props is now available for us to consume. Our StudentList component should now look like this:

```
import React, {Component} from 'react'
import {connect} from 'react-redux'

class StudentList extends Component{
  render(){
      const students = this.props.students.map(student=>{
        return(
          <tr key={student.id}>
            <td>{student.id}</td>
            <td>{student.name}</td>
            <td>{student.score}</td>
            <td>{student.nationality}</td>
          </tr>
        )
      })
    return(
      <div>
        <h2>List of students</h2>
        <table>
          <tbody>
```

```
            {students}
          </tbody>
        </table>
      </div>
    )
  }
}
const mapStateToProps = (state)=>{
  return{
    students: state.studentReducer.students
  }
}
export default connect(mapStateToProps)(StudentList)
```

Map Dispatch to Props

We have seen how easy it is to combine Redux and React. We were able to pass state from Redux to React as Props. Now, we will take it a step further by mapping dispatch to props

Let's update our studentList component:

1. To map dispatch to props, add the following lines of codes after mapStateToProps function

```
const mapDispatchToProps = (dispatch)=>{
  return{
    deleteStudent: (id)=>{dispatch({type:'DELETE_STUDENT', id:id})}
  }
}
```

2. Replace

export default connect(mapStateToProps)(MapDisPatchToProps);

with

export default connect(mapStateToProps, mapDispatchToProps)(StudentList);

3. Create a handleClick function

```
handleClick = (id)=>{
    this.props.deleteStudent(id)
  }
```

4. Inside the map function, after

```
<td>{student.nationality}</td>
add this
<td><button onClick={()=>{this.handleClick(student.id)}}>Delete</button></td>
```