Framework Training
React
London July 2018
Exercise N-redux

- This exercise uses the **Redux** state management pattern.
- Install and run the starter project.

```
npm install
npm start
```

- This project was built with create-react-app.
- Then the **redux** and **react-redux** packages were installed with npm.

Redux

- **Redux is a software design pattern** which aims to isolate the state of your application into one separate **store**.
- The store object lives separate from the hierarchy of React components.
- The components communicate indirectly with the store.
- This approach avoids the complexity of passing information between nested components using props.
- Redux defines a **store** where the application state is held. Our application will hold two things, the name of the airport and the star rating of the hotel.

```
{
airport : "gatwick",
hotel : "four",
}
```

- When components want to notify the store of a change, they **dispatch actions**.
- An action contains a type and some data.
- Only two types of action are needed in this example.

```
{type:AIRPORT, data:"Gatwick"}
{type:HOTEL, data:"four"}
```

- The type properties are strings defined as constants.

```
const AIRPORT = "airport" ;
const HOTEL = "hotel" ;
```

- When an action is dispatched, it is sent to a **reducer** function.
- The reducer is a custom function that you write which describes how the store should change when a specific action happens.
- The reducer does not directly change the store.
- It returns a new updated copy of the state, which the store updates.
- Whenever the store updates itself, it then notifies any **subscribers** that it has changed.

React-Redux

- **React-Redux** is a library which implements the Redux pattern in React.
- The starter version of this project has installed these libraries:

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

Implementing Redux

- Create a **redux folder** in the src folder.
- Define two constants which are the types of action that will be dispatched in new file **redux/actions.js**

```
const AIRPORT    = "airport" ;
const HOTEL      = "hotel" ;

export { AIRPORT, HOTEL } ;
```

- Define a minimal reducer function in new file **redux/reducer.js**

```
const DEFAULT_STATE = {
    airport : "gatwick",
    hotel : "four",
}

export let reducer = (state=DEFAULT_STATE, action) => {
    switch (action.type) {
        default:
            return state;
    }
}
```

- Create a file **redux/store.js** which defines the Redux store.

```
import { createStore } from 'redux';
import { reducer } from "./reducer";
```

```
let store = createStore( reducer );

export { store }
```

Redux web tools

- Install the **Redux WebTools** as a Chrome browser extension
- This will aid debugging of the React Redux app.

```
https://chrome.google.com/webstore/detail/redux-
devtools/lmhkpmbekcpmknklioeibfkpmmfibljd
```

- We need to add a 2nd parameter to the createStore function in redux/store.js in order to communicate with the Redux WebTools.

```
let store = createStore(reducer,
window.__REDUX_DEVTOOLS_EXTENSION__());
```

Test Redux

- In src/index.js add code to test dispatching an action to the store.
- Import the store and the AIRPORT constant.

```
import { store } from './redux/store';
import { AIRPORT } from './redux/actions';
```

- Dispatch a test action to the store.
- This code is wrapped in a two second time delay.

```
window.setTimeout( () => store.dispatch( { type:AIRPORT,
data:"heathrow"}) , 2000 );
```

- Open the Redux-WebTools to confirm if this works.

Dispatching actions from components

- We want to adapt our Airport and Hotel components to communicate with the Redux store.
- They need to **send** it **actions**.
- They need to **receive notification** when the store has changed.
- We will define two functions.
- **mapDispatchToProps** is a function which defines an event handler that dispatches actions.
- **mapStateToProps** is a function that contains the parts of the store object that a component needs to know about.

- React-Redux includes a **connect** function.
- This higher-order function takes mapDispatchToProps and mapStateToProps as its two arguments.
- It returns a new function.
- We then invoke the new function passing it the Airport or Hotel component as an argument.
- It then returns a new enhanced component which can communicate with the Redux store via its props.
- Open **components/airport.js**
- Import the connect function and the AIRPORT constant.

```
import { connect } from 'react-redux';
import { AIRPORT } from '../redux/actions';
```

- Define a **mapStateToProps** function

```
function mapStateToProps( state ) {
    return {
        airport : state.airport
    }
}
```

- Define a **mapDispatchToProps** function.
- This creates a new event-handler which dispatches actions.

```
function mapDispatchToProps( dispatch ) {
    return {
        setAirport( a ) {
            dispatch( { type:AIRPORT, data:a } )
        }
    }
}
```

- Wire up the two radio buttons to call this event-handler when they are selected.

```
onChange={ () => setAirport('Gatwick')}
onChange={ () => setAirport('Heathrow')}
```

- The function setAirport is passed into this component as a prop. Use **destructuring** to gain access to it.

```
let {setAirport} = this.props;
```

- Create a new component using **connect**.

```
let AirportR = connect( mapStateToProps, mapDispatchToProps ) (
Airport ) ;

export { AirportR }
```

- In index.js change the import statement and instance.

```
import { AirportR } from './components/airport';
<AirportR />
```

- In its current state, the run-time error occurs.
- To pass the Redux store between all components implicitly the top-level view is wrapped in a **Provider**.

```
import { Provider } from "react-redux";

<Provider store={store}> .. </Provider>
```

- Test the application: clicking the Gatwick or Heathrow radio buttons should dispatch actions to the store.

Add custom logic to the reducer

- Add a case statement to the reducer.
- This describes how the store should be updated when an AIRPORT action is dispatched.
- Note the reducer is a pure function. It does not directly change the store.
- Instead it makes a new copy of the state object, and modifies the copy.
- Redux will then update the store with this copy.
- This code shows two approaches using Object.assign and the spread operator to make a separate object.

```
        case AIRPORT:
        return { ...state , airport: action.data };
        // return Object.assign( {} , state, { airport:
action.data });
```

Hotel component

- Implement the Hotel component using the same approach.
- Here are some code fragments which need to be added to multiple files.

```
import { connect } from 'react-redux';
import { AIRPORT } from '../redux/actions';
```

```
function mapStateToProps() {}
function mapDispatchToProps() {}

onChange={ () => setHotel('four')}
onChange={ () => setHotel('five')}

let {setHotel} = this.props;

let HotelR = connect( mapStateToProps, mapDispatchToProps ) (
HotelR ) ;

export { HotelR }

import { HotelR } from './components/hotel';
<HotelR />

case HOTEL:
return { ...state , hotel: action.data };
// return Object.assign( {} , state, { hotel: action.data });
```

Holiday component

- The Holiday component should display a sentence which updates as the state changes.
- Use connect to adapt the component to talk to the Redux store.

```
function mapStateToProps( state ) {
    return {
        hotel : state.hotel,
        airport : state.airport
    }
}

function mapDispatchToProps( dispatch ) {
    return {}
}

let HolidayR = connect( mapStateToProps, mapDispatchToProps ) (
Holiday ) ;

export { HolidayR }
```

- Note this component does not dispatch any actions.
- The state is passed in as props to the view.

```
    let { hotel, airport } = this.props;
    <p>{ hotel } { airport }</p>
```

- Update the new HolidayR name in trips.js

```
import { HotelR } from './components/hotel';
<HolidayR/>
```

Review the application in Redux WebTools

- Note that changes of state cause new props to be sent to the HolidayR component.
- Add a lifecycle method in this component to log this.

```
componentWillReceiveProps( nextProps ) {
    console.log( "componentWillReceiveProps" ,
nextProps,"*",this.props );
}
```

Reset button

- Create a new Reset component which dispatches a RESET action and sets the store back to its default values.
- Define and export a new constant in **actions.js.**

```
const RESET = "reset" ;
```

- Add a new case to the **reducer**.

```
case RESET:
return DEFAULT_STATE;
```

- Create a new component which consists of a button that calls a reset method.

```
class Reset extends Component {
  render() {
      let {reset} = this.props;
      return (
          <section className="button" onClick=
{reset}>Reset</section>
      )
  }
}
```

- Add functions to dispatch a RESET action to the store.

```
    function mapStateToProps( state ) {
        return {}
    }

    function mapDispatchToProps( dispatch ) {
        return {
            reset() {
                dispatch( { type:RESET } )
            }
        }
    }

    let ResetR = connect( mapStateToProps, mapDispatchToProps )
( Reset ) ;

    export { ResetR }
```

- Import the new component into index.js and create an instance.

```
import { ResetR } from './components/reset';
<ResetR/>
```

- Clicking the reset button should restore the default state of the store and reset the UI in the Airport/Hotel components.
- The Redux pattern means that we can restructure our components and the application continues to work. Move ResetR inside Trips to test this concept.