strategy | consulting | digital | technology | operations

REACTJS – MODULE 10

REACT-REDUX

accenture

AGENDA

- Introduction to redux
- Basic principles of Redux
- Redux flow
- How to use redux with respect to react components
- Demo
- Activity

accenture

Introduction to Redux

Some time in react managing state of a component can become complex task, as it involves multiple event handlers to be created for every state update

Instead of doing this every time we can create a separate module which can keep track of all the data

Redux supports this concept of keeping all the data/state in one single place. It also supports for modifying data

Redux helps to use same data in multiple places, which also enables inter component communication

accenture

Basic Principles Of Redux

Single Immutable store

State/Data inside store is immutable, means non updatable

Actions invoke changes

User actions are captured which will trigger the state change

Reducer responsible to update store

Receives the action and current state and return updated state

Store

In redux architecture stores are the area where we can keep data/state of an application

There will be only one store for entire application and state inside store is immutable

The state of a store can be modified only with the help of reducers

accenture

Action

In an react application there can be many actions which can be created

Actions are captured by react component and notify to reducer about the state change request

Example

When the user clicks on “create user” button the corresponding action is triggered which will indicate to reducer that new user must be created with new state

accenture

Reducer

Reducers are pure functions which accepts arguments and returns value

In Redux architecture reducer will take current state and action as an argument and returns the modified value(state) which will update the store

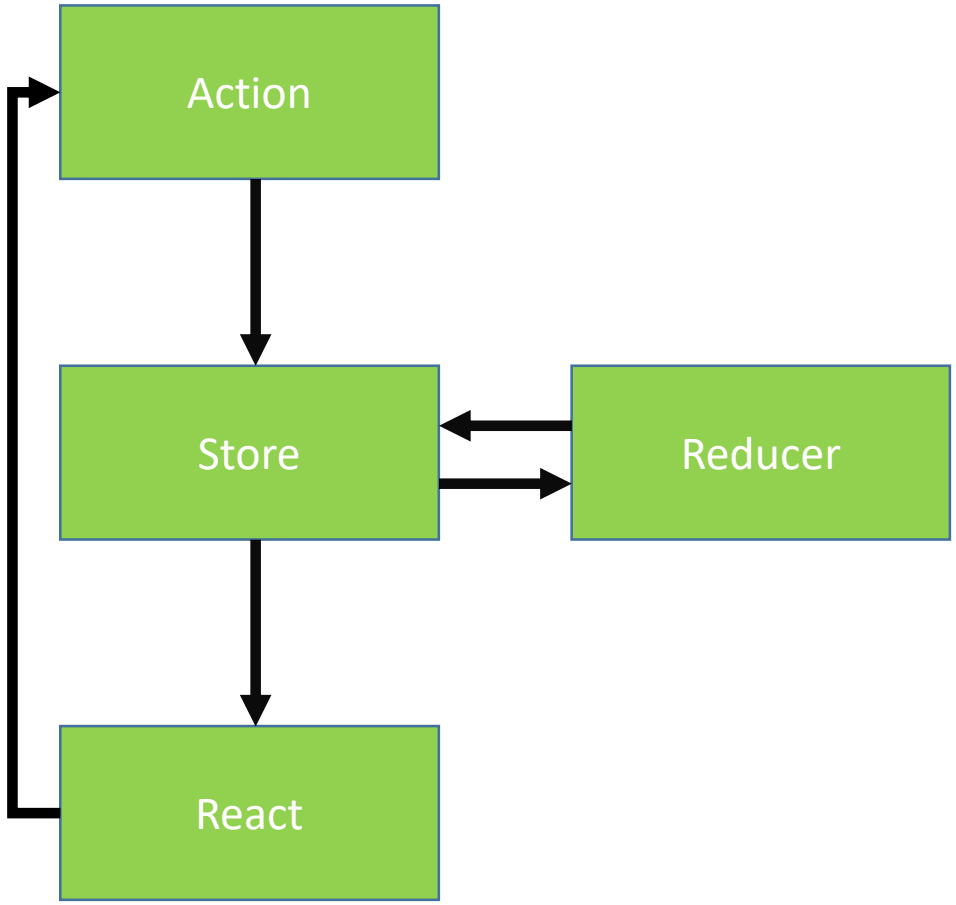
There can be multiple reducers in an single application. Multiple reducers can also handle single action

Reducers in redux are very simple and approachable

All reducers are called when an action is dispatched but each reducer will handle only its slice of state

accenture

Redux flow



Actions are used to trigger state change request. The request will come from react components

Reducers receives an action and current state as arguments and updates the new state in the store

Store gets notified about the new state change and further notifies to all react components which are connected to store

React-redux will identify whether the react component requires re-rendering with the changes or not

Finally react components gets rendered and UI gets updated

Difference between Flux and Redux

Flux	Redux
Stores in flux will have state and also will contain change logic	Stores in Redux will have state but change logic is kept separate
Can have multiple stores	Will have Single store
Mutable state	Immutable state
Singleton Dispatcher	No dispatcher instead multiple reducers can exist

accenture

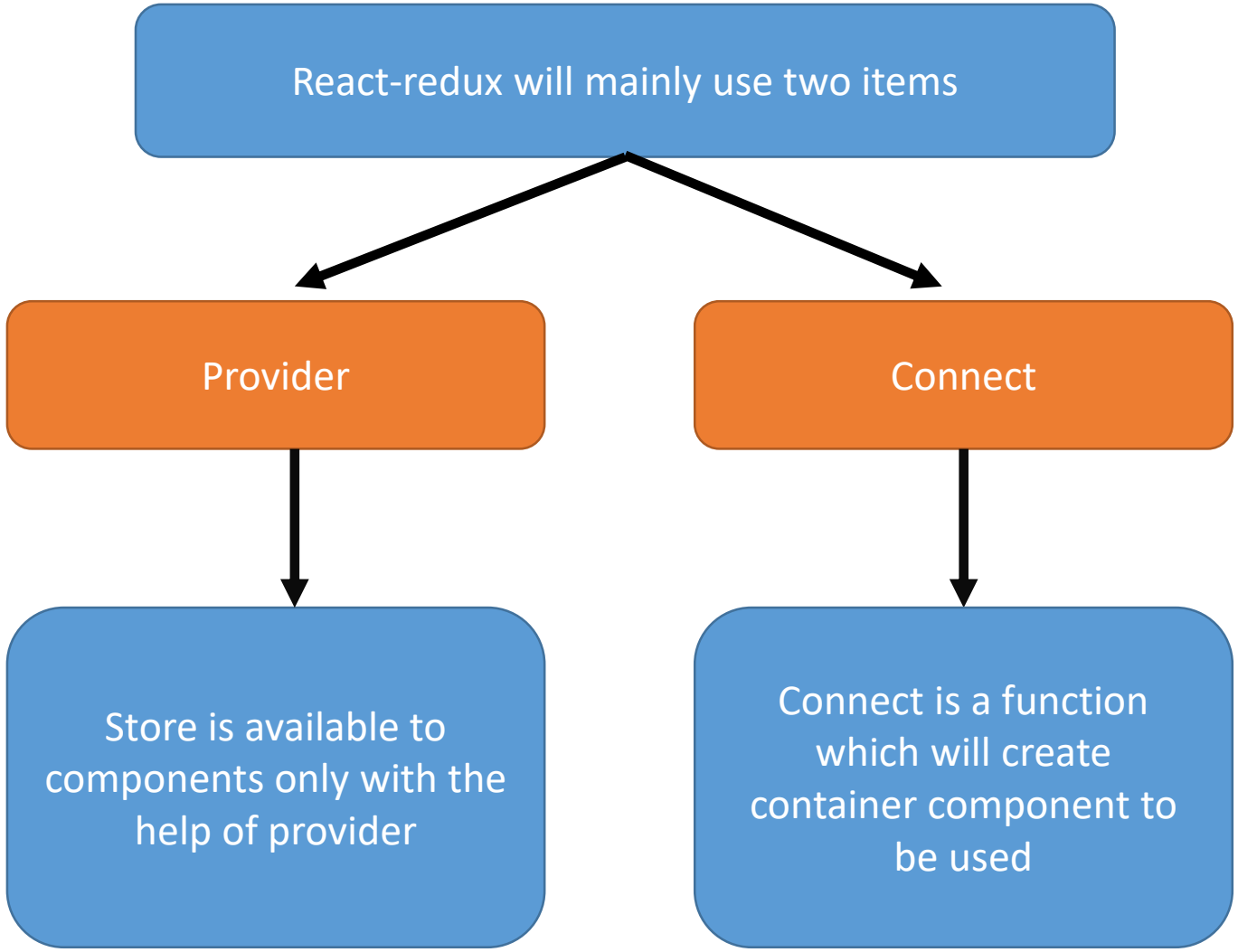
How to connect React with Redux

Since redux is a library which will allow to keep track of complete applications state/data in a single place, hence this can be used with other libraries or framework

Example
Redux can be used with React library
Redux can be used with Angular framework

In this module we will be focusing on how react can be connected with redux and utilize its features using react-redux library

How to connect React with Redux (Cont...)



Provider

```
<Provider store={this.props.store}>
    <App/>
</Provider>
```

In the above code snippet provider is connecting top level component of an application that is “App” component to the store, this way the store is available to all other components within that application

Connect

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

Connect is a function which wraps the react component(container component) so its connected to redux store. With this we can declare what parts of the store we would like to attach to react component as props and also declare what actions will be exposed as props

mapStateToProps is a function which will identify which states will be exposed as props on the container component

mapDispatchToProps is a function which will identify which actions will be exposed as props on the container component

accenture

Demo – React Redux

Create new project reduxexample and install redux

```
C:\reactdemos\reduxexample>npm install redux react-redux
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.2 (node_modules\react-scripts\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.2: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.3 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.3: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

+ redux@3.7.2
+ react-redux@5.0.6
added 235 packages in 134.405s
```

accenture

Demo – React Redux – Create Action

Create new folder “actions” inside src folder of a project. Create courseActions.js inside actions folder

```
export function createCourse(course) {
    return {type:'CREATE_COURSE',course:course};
}
```

Demo – React Redux – Create Reducer

Create new folder “reducers” inside src folder of a project. Create courseReducers.js inside reducers folder

```
export default function courseReducers(state=[],action) {
switch (action.type){
    case 'CREATE_COURSE':
        return [...state,Object.assign({},action.course)];
    default:
        return state;
}
}
```

Demo – React Redux – Create Reducer (Cont...)

Redux architecture can use multiple reducers per application. In order to keep track of all reducers in a single file, we can create index.js file inside reducers folder

```
import {combineReducers} from 'redux';
import courses from './courseReducers';
const rootReducer=combineReducers({
    courses
});
export default rootReducer;
```

Demo – React Redux – Create Store

Create new folder “store” inside src folder of a project. Create configureStore.js inside store folder

```
import {createStore} from 'redux';
import rootReducer from '../reducers';
export default function configureStore(initialState) {
    return createStore(
        rootReducer,
        initialState

    );
}
```

Demo – React Redux – Create View

Create CoursesPage.js inside src folder

```
import React,{Component} from 'react';
import {connect} from 'react-redux';
import * as courseActions from './actions/courseActions';
class CoursesPage extends Component{
    constructor(){
        super();
        this.state={
            course:{title:''}
        };
        this.onTitleChange=this.onTitleChange.bind(this);
        this.onClickSave=this.onClickSave.bind(this);

    }
```

Demo – React Redux – Create View (Cont ...)

```
    onTitleChange(event){
        const course=this.state.course;
        course.title=event.target.value;
        this.setState({
            course:course
        });
    }

    onClickSave(event){
        this.props.createCourse(this.state.course);
    }
    courseRow(course,index){
        return <div key={index}>{course.title}</div>;
    }
```

Demo – React Redux – Create View (Cont ...)

```
    render(){
        return(
            <div>
                <h1>Course page</h1>
                {this.props.courses.map(this.courseRow)}
                <h2>Add Course</h2>
                <input type="text" onChange={this.onTitleChange}
                       value={this.state.course.title}/>
                <input type="submit" value="Save" onClick={this.onClickSave}/>
            </div>
        );
    }
}
```

Demo – React Redux – Create View (Cont ...)

```
function mapStateToProps(state) {
    return{
        courses:state.courses
    }
}
function mapDispatchToProps(dispatch) {
    return{
        createCourse:course=>dispatch(courseActions.createCourse(course))
    }
}
export default connect(mapStateToProps,mapDispatchToProps)(CoursesPage);
```

Demo – React Redux – Provider (index.js)

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import configureStore from './store/configureStore';
import {Provider} from 'react-redux';
const store=configureStore();
ReactDOM.render(<Provider store={store}>
    <App/>
</Provider>, document.getElementById('root'));
registerServiceWorker();
```

accenture

Demo – React Redux – App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import CoursesPage from './CoursesPage';
class App extends Component {
  render() {
    return (
      <div>
        <CoursesPage/>
      </div>
    );
  }
}
export default App;
```

Demo – React Redux – Explanation

```
export function createCourse(course) {
    return {type:'CREATE_COURSE',course:course};
}
```

CREATE_COURSE is name of the action which will be triggered when the user clicks on save button from the view

"course" is a data which is captured from the textbox

Demo – React Redux – Explanation (Cont ...)

```
export default function courseReducers(state=[],action) {
switch (action.type){
    case 'CREATE_COURSE':
        return [...state,Object.assign({},action.course)];
    default:
        return state;
}
}
```

Create course action will notify the course reducer, and reducer will add new course

Demo – React Redux – Explanation (Cont ...)

```
import {combineReducers} from 'redux';
import courses from './courseReducers';
const rootReducer=combineReducers({
    courses
});
export default rootReducer;
```

rootReducer is a place where we can keep list of all reducers which are used inside the application

Demo – React Redux – Explanation (Cont …)

```
import {createStore} from 'redux';
import rootReducer from '../reducers';
export default function configureStore(initialState) {
    return createStore(
        rootReducer,
        initialState

    );
}
```

Store will contain reducer information, because when the reducer updates the data, store should get notification. Also store will receive initial state from reducer

Demo – React Redux – Explanation (Cont ...)

```
import React,{Component} from 'react';
import {connect} from 'react-redux';
import * as courseActions from './actions/courseActions';
class CoursesPage extends Component{
    constructor(){
        super();
        this.state={
            course:{title:''}
        };
        this.onTitleChange=this.onTitleChange.bind(this);
        this.onClickSave=this.onClickSave.bind(this);

    }
```

The page have course object and title as its one of the property which is initially empty

Demo – React Redux – Explanation (Cont ...)

```
onTitleChange(event){
    const course=this.state.course;
    course.title=event.target.value;
    this.setState({
        course:course
    });
}

onClickSave(event){
    this.props.createCourse(this.state.course);
}
courseRow(course,index){
    return <div key={index}>{course.title}</div>;
}
```

When the user enters course name in the textbox the new value is captured and set the local state

When clicked on save button the corresponding action is triggered

Demo – React Redux – Explanation (Cont ...)

```
render(){
    return(
        <div>
            <h1>Course page</h1>
            {this.props.courses.map(this.courseRow)}
            <h2>Add Course</h2>
            <input type="text" onChange={this.onTitleChange}
                value={this.state.course.title}/>
            <input type="submit" value="Save" onClick={this.onClickSave}/>
        </div>
    );
}
}
```

This will display all the list of courses, this data is retrieved from the store

Demo – React Redux – Explanation (Cont ...)

```
onTitleChange(event){
    const course=this.state.course;
    course.title=event.target.value;
    this.setState({
        course:course
    });
}

onClickSave(event){
    this.props.createCourse(this.state.course);
}
courseRow(course,index){
    return <div key={index}>{course.title}</div>;
}
```

Display all courses one by one

Demo – React Redux – Explanation (Cont …)

```
function mapStateToProps(state) {
    return{
        courses:state.courses
    }
}
function mapDispatchToProps(dispatch) {
    return{
        createCourse:course=>dispatch(courseActions.createCourse(course))
    }
}
export default connect(mapStateToProps,mapDispatchToProps)(CoursesPage);
```

Retrieves all the course name from the store

Identifies what actions are available in component which can be used to notify reducer

Demo – React Redux – Explanation (Cont …)

```
function mapStateToProps(state) {
    return{
        courses:state.courses
    }
}
function mapDispatchToProps(dispatch) {
    return{
        createCourse:course=>dispatch(courseActions.createCourse(course))
    }
}
export default connect(mapStateToProps,mapDispatchToProps)(CoursesPage);
```

Connect function used to interact with redux. Here CoursePage is a container component to which redux will send the data

Demo – React Redux – Explanation (Cont …)

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import configureStore from './store/configureStore';
import {Provider} from 'react-redux';
const store=configureStore();
ReactDOM.render(<Provider store={store}>
    <App/>
</Provider>, document.getElementById('root'));
registerServiceWorker();
```

Provider makes store available to App component

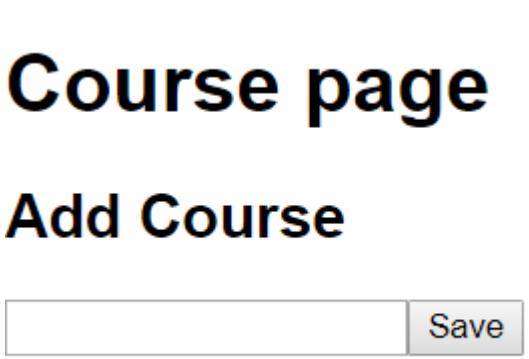
Demo – React Redux – Explanation (Cont ...)

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import CoursesPage from './CoursesPage';
class App extends Component {
  render() {
    return (
      <div>
        <CoursesPage/>
      </div>
    );
  }
}
export default App;
```

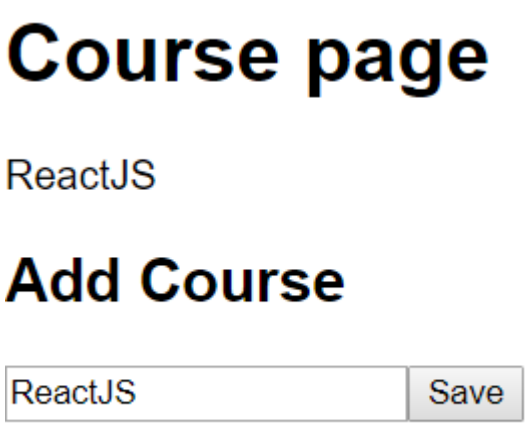
CoursePage is displayed

Demo – React Redux

Output



Output after clicking on save course



Activity

1. Use existing application “reduxexample” which is created using react with redux architecture
2. Create a view which will have another textbox. In this textbox a user will enter course fee of the course
3. When the user clicks on save button, course name and course fee should be saved
4. The view should display updated list of course and corresponding course fees

Activity – Expected Output

Output

Output after clicking on save course

Redux - Router

Whenever there is a requirement to achieve routing with react-redux application, then one can utilize react's router concept combined with redux concept

Redux is source of truth for applications state however router is source of truth for URL inside an application

Combing redux with router is optional implementation inside an application and completely depends on the need

How To Implement Redux - Router

In order to combine redux with router the first step is to install necessary libraries inside the application
npm install redux react-redux
npm install react-router-dom

Once the libraries are installed, next is to provide the Router and Route information. These details will be given inside the provider

```
<Provider store={store}>
  <Router>
     <Route path="/Home" component={Home}/>
  </Router>
</Provider>
```

accenture

How To Implement Redux - Router

Finally the view must be provided with the help of Link component. Using this a user can see the respective links on the user interface

```
<Link to="/Home">Home</Link>
```

MODULE SUMMARY

- Understand why redux is important
- Understand how redux works
- Understand
  - Store
  - Reducer
  - Action
  - Provider

accenture

THANK YOU

accenture