Lab 6 - Rest API & React Redux

Node - Rest API

Rest API เป็นการมองการทำงาน services (หรือ APIs) ต่าง ๆ ในรูปแบบของ Resource และ ใช้ HTTP Verbs (GET, POST, PUT, DELETE) ในการจัดการกับ Resource นั้น ๆ

ตัวอย่างโครงสร้าง Bear.js นี้ มีการใช้ express, body-parser, router และ cors โดยมีหน้าที่ดังนี้

- Router ใช้สำหรับจัดการ route ของ path ที่ร้องขอจาก client มายัง backend รวมถึงการกำหนด middleware ต่าง ๆ ผ่าน app.use ก่อนนำไปใช้งาน นอกจากนี้ ยังสามารถกำหนดให้ทุก ๆ การร้องขอเรียกผ่าน /api/ ได้อีกด้วย
- cors ใช้สำหรับรองรับการเรียกใช้ client จากคนละ domain
- app.use(“*”,...) หมายถึง path นอกเหนือจากที่กำหนด จำเป็นต้องวางไว้สุดท้ายก่อนจะเปิดพอร์ท รอรับการเชื่อมต่อ ในกรณีคือ กรณีที่ร้องขอ path ที่ไม่ได้กำหนดไว้ ก็จะให้ส่งข้อความว่า 404 Not found

Bear.js

```
let express = require('express');
let bodyParser = require('body-parser');
let router = express.Router();
let cors = require('cors');
let app = express();
app.use(cors());

// all of our routes will be prefixed with /api
app.use('/api', bodyParser.json(), router);   //[use json]
app.use('/api', bodyParser.urlencoded({ extended: false }), router);

let bears = [{'id':0,'name':'pooh','weight': 211},
   {'id':1, 'name':'vinnie','weight': 111}
];

router.route('/bears').get((req, res) =>  res.json(bears) );

app.use("*", (req,res) => res.status(404).send('404 Not found') );
app.listen(80,  () => console.log("Server is running") );
```

ทดสอบการเรียกใชงานโดยเปิด browser ไปที่ http://localhost:8000/api/bears

Output:

```
[
            {
                        id: 0,
                        name: "pooh",
                        weight: 211
            },
            {
                        id: 1,
                        name: "vinnie",
                        weight: 111
            }
]
```

Create/Read/Update/Delete (CRUD)

Create

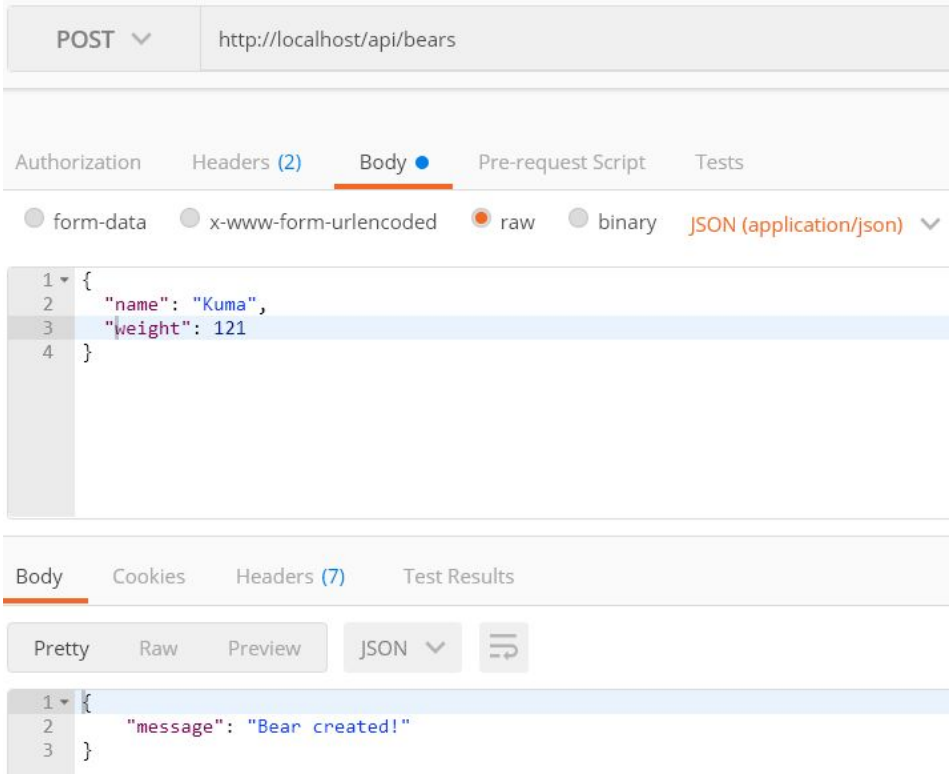
```
...
let bears = [{'id':0,'name':'pooh','weight': 211},
   {'id':1, 'name':'vinnie','weight': 111}
];

let bearIndex=2;

router.route('/bears')
   // get all bears
   .get( (req, res) =>  res.json(bears) )

   // insert a new bear
   .post( (req, res)=> {
       var bear = {};
       bear.id =  bearIndex++;
       bear.name = req.body.name
       bear.weight = req.body.weight
       bears.push(bear);
       res.json( {message: 'Bear created!'} )
   })
...
```

ใช้โปรแกรม Postman ในการทดสอบ method POST เพื่อสร้าง bear มาใหม่ ดังนี้ เลือก method POST, Body เป็น raw แบบ JSON และกำหนดข้อมูลในฟอร์ม



Read

หลังจากสร้าง bear แล้ว ก็ลองอ่านค่าดูอีกครั้งว่าสร้างสำเร็จหรือไม่

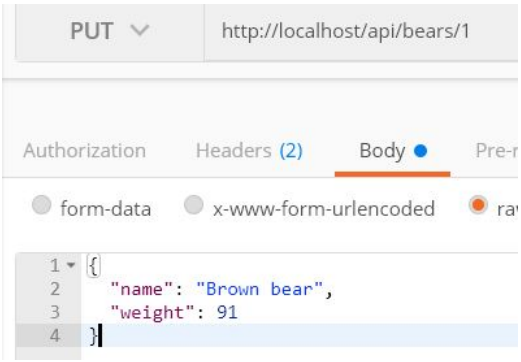
Update and Delete

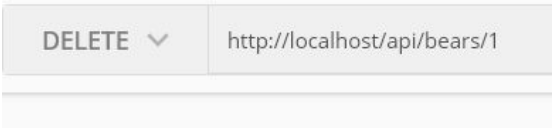
```
..

router.route('/bears/:bear_id')
    .get ( (req,res) => res.json(bears[req.params.bear_id]))   // get a bear

    .put ( (req,res) => {                                     // Update a bear
        var id = req.params.bear_id
        bears[id].name = req.body.name;
        bears[id].weight = req.body.weight;
        res.json({ message: 'Bear updated!' + req.params.bear_id});
    })

    .delete ( (req,res) => {                          // Delete a bear
        delete     bears[req.params.bear_id]
        res.json({ message: 'Bear deleted: ' + req.params.bear_id});
    })
...
```





หลังจาก update bear ผ่าน PUT โดยใช้ POSTMAN แล้วทดลองเรียก http://localhost/api/bears อีกครั้ง และลองเรียก DELETE และ สังเกตผลลัพธ์ที่ได้


```
[
  - {
        id: "5935512001",
        name: "Somchai",
        surname: "Khemklad",
        Major: "CoE",
        GPA: 3.32
    },
  - {
        id: "5935512002",
        name: "Yaya",
        surname: "Rakngam",
        Major: "SE",
        GPA: 3
    }
]
```


5.1 จงAPI ที่สามารถทำ CRUD กับ JSON ต่อไปนี้ พร้อมตั้งชื่อตัวแปร ฟังก์ชัน URL ให้เหมาะสมกับการทำงาน

Redux

Redux คือการจัดการ state โดยการรวม state ทั้งหมดในลักษณะ Global ไว้ที่ store ที่เดียว ที่ ทุก ๆ คอมโพเน้นท์สามารถเรียกใช้งานในผ่าน props และ การจัดการ state ทำผ่าน action ที่กำหนดชนิดของการทำงาน และ จัดการข้อมูลใหม่ผ่าน reducer (หรือจะมองว่าเป็น controller ก็ได้) ก่อนนำข้อมูลใหม่ที่ได้ไปเก็บไว้ใน store

Review Counter react app

สร้าง react app ด้วย `npx create-react-app myapp` จากนั้น สร้างโปรแกรม Counter เหมือนใน Lab ที่ 4 ดังนี้

Count.js

```
class Count extends Component {
   state = {number: 0}
   add = () => this.setState({number: this.state.number + 1})
   minus = () => this.setState({number: this.state.number - 1})

   render() {
       return (
               <div style={{margin: '20px'}}>
                   Counter: {this.state.number} <br/>
                   <button onClick={this.add} > +</button>
                   <button onClick={this.minus}> -</button>
               </div>
       );
   }
}

export default Count
```

Counter: 0

+ -

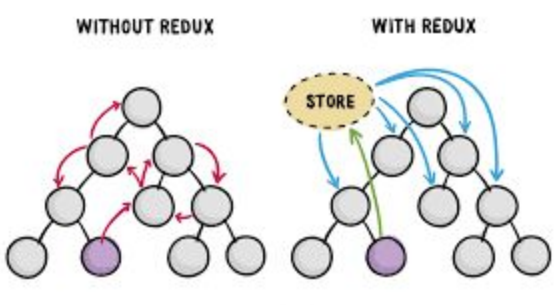
App.js

```
import React, {Component} from 'react';
import Count from './Count'

class App extends Component {
   render() {
       return <Count />
   }
}
export default App
```

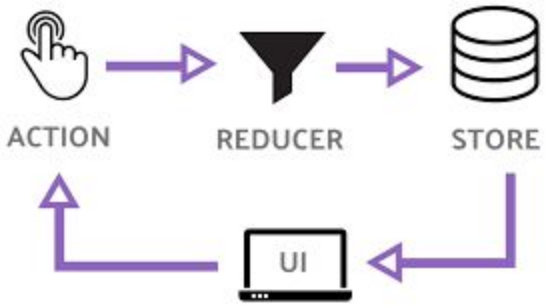
Why Redux

ถ้าไม่ใช้ Redux เราจำเป็นจะต้องส่ง state จากแม่สู่ลูกไล่ตามลำดับชั้น ไม่สามารถข้ามชั้นได้ และ ลูกก็ไม่สามารถ



เปลี่ยนแปลงค่า state โดยตรงของ component อื่น ๆ ได้ ต้องทำผ่าน function ที่อยู่ใน component ของแม่ ทำให้การจัดการทำได้ยุ่งยาก Redux มีแนวคิดคือรวมทุกอย่างไว้ที่ store กลาง และ ทุก ๆ component สามารถเข้าถึง store นั้นได้อย่างอิสระ

การเปลี่ยนแปลงค่าใน store จะทำผ่าน action (โดยการ dispatch) และส่งไป reducer เพื่อนำ state ใหม่เข้าสู่ store เมื่อ state ใน store มีการเปลี่ยนค่าใหม่ ก็จะส่งผลให้ UI ปรับปรุงหน้า component นั้น



Change to Redux

ต่อไปจะเริ่มเปลี่ยนจากตัวอย่างข้างบน โดยใช้ redux ดังนี้

1. yarn add redux react-redux (หรือจะใช้ npm i --save redux react-redux ก็ได้)
2. นำ <Provider> คลุม tag root div ไว้ (เพื่อให้เข้าถึง store ได้)
3. สร้าง action เป็น dispatcher เพื่อระบุ type ให้ reducer (controller) ทราบว่าต้องการให้ทำอะไรการสั่ง
4. สร้าง reducer เพื่อทำงานตาม action ที่ได้รับมา แล้ว ส่งค่าเก็บใน store

Provider, Store, Action, Reducer

App.js

```
import React, {Component} from 'react';
import {Provider} from 'react-redux'
import {createStore, combineReducers} from 'redux'
import Count from './Count'

// ========  action (As Dispatcher) ==========
export const add = () => ({type: 'ADD'})
export const minus = () => ({ type: 'MINUS'})

// ========  reducer (As Controller) =========
export const numberReducer = (state = 0, action) => {
   switch (action.type) {
       case 'ADD':
           return state + 1
       case 'MINUS':
           return state - 1
       default:
           return state
   }
}
export const rootReducer = combineReducers({number: numberReducer})
export const store = createStore(rootReducer)
```

```
// ======== wrap root element by Provider with Store ========
class App extends Component {
   render() {
       return <Provider store={store}><Count/> </Provider>
   }
}

export default App
```

สามารถลอง สั่ง dispatch แล้ว getState ที่อยู่ใน store มาดูได้

```
store.dispatch(add());
console.log('getState: ', store.getState());
```

mapStateToProps

Count.js

เรียกใช้ dispatch จาก store เพื่อ ให้ action ตามที่ผู้ใช้สั่ง และอ่านค่า state ที่เก็บใน store แล้ว map เข้ากับ props โดยการ mapStateToProps

```
import React, {Component} from 'react';
import './App.css';
import { add,minus, store} from './App'
import { connect } from 'react-redux'

class Count extends Component {

   render() {
       return (
               <div style={{margin: '20px'}}>
                   Counter: {this.props.number} <br/>
                   <button onClick={() => store.dispatch(add())}>+</button>
                   <button onClick={() => store.dispatch(minus())}>-</button>
               </div>
       );
   }
}

const mapStateToProps = (state) => {
   return { number: state.number }
}

export default connect(mapStateToProps)(Count);
```

dispatch action จะต่างจากการเรียก function call ธรรมดา ตรงที่ return object ที่ได้มาจะถูกส่งต่อให้ reducer เพื่อนำค่าที่ได้มาจัดการและเก็บ global state ใน store ซึ่งจะมีผลทำให้ render() ถูกเรียกทำงานใหม่ เมื่อ state มีการเปลี่ยนค่าใหม่

mapDispatchToProps

หากกรณีที่ต้องการเรียก action ผ่าน props โดยการ mapDispatchToProps ดังนี้

Count.js

```
import React, {Component} from 'react';
import { add,minus, store} from './App'
import { connect } from 'react-redux'

class Count extends Component {
   render() {
       return (
               <div style={{margin: '20px'}}>
                   Counter: {this.props.number} <br/>
                   <button onClick={ this.props.add}>+</button>
                   <button onClick={ this.props.minus}>-</button>
               </div>
       );
   }
}

const mapStateToProps = (state) => {
   return { number: state.number }
}

const mapDispatchToProps = (dispatch) => {
   return {
       add : ()=>dispatch(add()),
       minus: ()=> dispatch(minus())
   }
}

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

Action with argument

หากต้องการเรียก action และส่ง argument ไปด้วย เช่นต้องการให้เพิ่มค่าทีละ 2 หรือ ตามจำนวนที่ส่งผ่าน argument ของฟังก์ชัน add สามารถทำได้ตามตัวอย่างนี้

```
// count.js
import { add,add2, minus, store} from './App'
 .....
<button onClick={() => store.dispatch(add2(2))}>+2</button>
 .....


// app.js
export const add2 = (number) => ({type: 'ADD2', number})
 .....
       case 'ADD2':
           return state + action.number
 .....
```

5.2 จงนำตัวอย่างการใช้ Redux ไปใช้กับโปรแกรม TodoList

Middleware: Redux Logger

สามารถใช้ middleware ReduxLogger เพื่อดูค่า state ใน store ได้สะดวก โดยมีขั้นตอนดังนี้

1. ติดตั้ง yarn add redux-logger
2. เพิ่ม code ใน App.js ดังตัวอย่าง

App.js

```
import logger from 'redux-logger'
...
export const store = createStore(rootReducer, applyMiddleware(logger))
```

ดูตัวอย่างจาก console จะเห็นว่า state จะแสดงขึ้นมาให้โดยอัตโนมัติ

Asynchronous action - Redux thunk

ในกรณีที่ action ที่ต้องการสั่งให้ทำมีการรอ เช่นการขอข้อมูลจาก API หรือ การเปิดไฟล์ เกิดการทำงานแบบ asynchronous action (จากตัวอย่างคือ function getBears) ที่มีการใช้ axios ไป query ออกมาผ่าน async function และ มีการใช้ await เพื่อรอการทำงาน ต้องมีการกำหนดการทำงานดังนี้

- ติดตั้ง axios และ redux-thunk
- เขียน async action
- กำหนด thunk ใน applyMiddleware เพื่อให้รอรับการทำงานแบบ asynchornous
- เพิ่ม reducer ใหม่ ที่ใช้รอรับ dispatch จาก async action

App.js

```
import React, {Component} from 'react';
import {Provider} from 'react-redux'
import {createStore, combineReducers, applyMiddleware } from 'redux'
import Count from './Count'
import Bear from './Bear'
import logger from 'redux-logger'
import axios from 'axios'
import thunk from 'redux-thunk'

// ========  action (As Dispatcher) ==========
```

```
export const add = () => ({type: 'ADD'})
export const add2 = (number) => ({type: 'ADD2', number})
export const minus = () => ({ type: 'MINUS'})

export const getBearsSuccess = bears => ({
   type: 'GET_BEARS_SUCCESS',
   bears
});
export const getBearsFailed = () => ({ type: 'GET_BEARS_FAILED'});

export const getBears = () => async (dispatch) => {
   try {
       console.log('get bear new')
       const response = await axios.get(`http://localhost/api/bears`)
       const responseBody = await response.data;
       console.log('response: ', responseBody)
       dispatch(getBearsSuccess(responseBody));
   } catch (error) {
       console.error(error);
       dispatch(getBearsFailed());
   }
}


// ========  reducer (As Controller) =========
export const numberReducer =  (state = 0, action) => {
   switch (action.type) {
       case 'ADD':
           console.log('my add...')
           return state + 1
       case 'MINUS':
           return state - 1
       case 'ADD2':
           return state + action.number
       default:
           return state
   }
}

export const bearReducer = (state = 0, action) => {
   switch (action.type) {
       case 'GET_BEARS_SUCCESS':
               console.log('action: ' , action.bears)
           return action.bears
       case 'GET_BEARS_FAILED':
               console.log('action: Failed')
           return action.bears
       default:
           return state
   }
}


export const rootReducer = combineReducers({number: numberReducer, bears:
bearReducer})
export const store = createStore(rootReducer, applyMiddleware(logger,thunk))


// ======== wrap root element by Provider with Store ========
```

```
class App extends Component {
   render() {
       return <Provider store={store}><Bear/><Count/> </Provider>
   }
}

export default App
```

Bear.js

```
import React, {Component} from 'react';
import {getBears} from './App'
import { connect } from 'react-redux'

class Bear extends Component {

   componentDidMount() {
       console.log('props',this.props)
       this.props.getBears()
   }

   renderBears = () => {
       if (this.props.bears) {
           return this.props.bears.map( (bear, index) => {
               console.log( bear.name)
               return (<li key={index}> {bear.name} : {bear.weight} </li>
           )})
       }
   }

   render() {
       return (
           <div style={{margin: '20px'}}>
               <h3>Render Bear</h3>
               <ul>
                   {this.renderBears()}
               </ul>
           </div>
       );
   }
}

const mapStateToProps =  ({bears}) => {
   return {bears}
}

const mapDispatchToProps = (dispatch) => {
   return {
       getBears: () => dispatch(getBears())
   }
}

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

5.3 จงนำ Redux ไปใช้กับ การดึงข้อมูลจาก Github API (Checkpoint)

5.4 จงนำ Redux ไปใช้กับ Insert / Delete / Update Bear ด้วย Redux (Homework)