Lab 6 - Rest API & React SWR

REST API - Node

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

Route	HTTP Verb	Description
/api/bears	GET	Get all the bears.
/api/bears	POST	Create a bear.
/api/bears/:bear_id	GET	Get a single bear.
/api/bears/:bear_id	PUT	Update a bear with new info.
/api/bears/:bear_id	DELETE	Delete a bear.

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 จำเป็นต้องวางไว้สุดท้ายก่อนจะเปิดพอร์ทรอรับการเชื่อมต่อ ในกรณีคือ กรณีที่ร้องขอ path ที่ไม่ได้กำหนดไว้ ก็จะให้ส่งข้อความว่า 404 Not found

เตรียม project

```
$ npm init --y
$ npm i -s express body-parser cors
```

Backend: 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 = {
   list: [
       { "id": 1, "name": "Winnie", "weight": 50 },
       { "id": 2, "name": "Pooh", "weight": 66 }]
}

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...'))
```

ทดสอบการเรียกใช้งานโดยเปิด browser ไปที่ http://localhost/api/bears

Output: (หากต้องการให้มีการจัดย่อหน้า ต้องติดตั้ง JSON Formatter chrome extension)

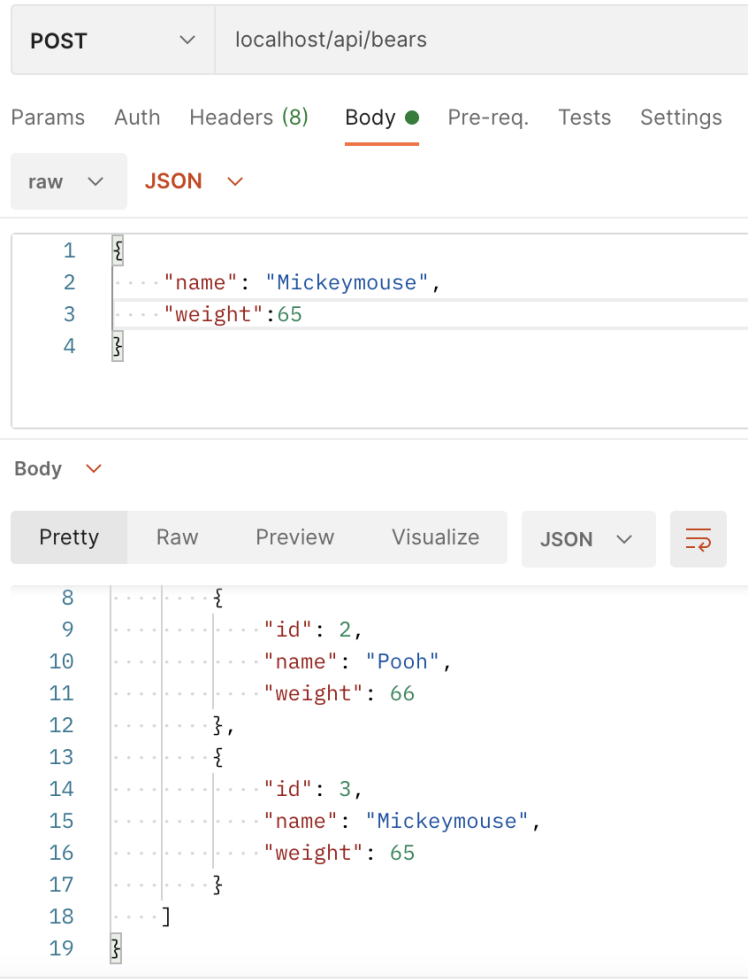
Create/Read/Update/Delete (CRUD)

Create

```
...
```

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

    .post((req, res) => {
        console.log(req.body)
        let newBear = {}
        newBear.id = (bears.list.length)?bears.list[bears.list.length - 1].id + 1:1
        newBear.name = req.body.name
        newBear.weight = req.body.weight
        bears = { "list": [...bears.list, newBear] }
        res.json(bears)
    })


...
```

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

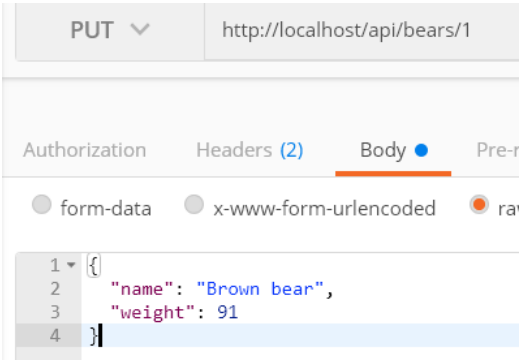


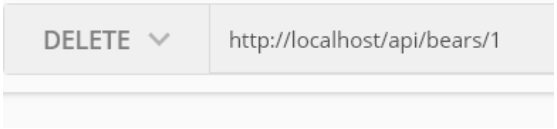
หมายเหตุ: หลังจากที่สร้าง bear สำเร็จ ในตัวนี้ return bears มาทั้งหมดเพื่อให้ง่ายในการตรวจสอบ แต่ก็มี trade-off กับการส่งข้อมูลกลับจาก server เมื่อใช้งานจริง อาจจะให้ส่งค่า json แสดงสถานะของการเพิ่ม bear เช่น ส่งค่า { message: “success” } แทน เป็นต้น

หลังจากสร้าง bear แล้ว ก็ลองอ่านค่าดูอีกครั้งว่าสร้างสำเร็จหรือไม่ โดยสามารถเปิดผ่าน http://localhost/api/bears

Read 1 bear, Update and Delete

```
..

router.route('/bears/:bear_id')
    .get((req, res) => {
        const bear_id = req.params.bear_id
        const id = bears.list.findIndex(item => +item.id === +bear_id)
        res.json(bears.list[id])
    })
    .put((req, res) => {
        const bear_id = req.params.bear_id
        const id = bears.list.findIndex(item => +item.id === +bear_id)
        bears.list[id].name = req.body.name
        bears.list[id].weight = req.body.weight
        res.json(bears.list[id])
    })

    .delete((req, res) => {
        const bear_id = req.params.bear_id
        console.log('bearId: ',bear_id)
        bears.list = bears.list.filter(item => +item.id !== +bear_id)
        res.json(bears.list)
    })

...
```





- **Note**: โปรแกรมยังไม่ได้ป้องกัน การ update/delete Bear ใน id ที่ไม่มีใน array

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

Complete backend code:

index.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 = {
   list: [
       { "id": 1, "name": "Winnie", "weight": 50 },
       { "id": 2, "name": "Pooh", "weight": 66 }]
}

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

   .post((req, res) => {
       console.log(req.body)
       let newBear = {}
       newBear.id = (bears.list.length)?bears.list[bears.list.length - 1].id + 1:1
       newBear.name = req.body.name
       newBear.weight = req.body.weight
       bears = { "list": [...bears.list, newBear] }
       res.json(bears)
   })

router.route('/bears/:bear_id')
   .get((req, res) => {
       const bear_id = req.params.bear_id
       const id = bears.list.findIndex(item => +item.id === +bear_id)
       res.json(bears.list[id])
   })
   .put((req, res) => {
       const bear_id = req.params.bear_id
       const id = bears.list.findIndex(item => +item.id === +bear_id)
       bears.list[id].name = req.body.name
       bears.list[id].weight = req.body.weight
       res.json(bears)
   })

   .delete((req, res) => {
       const bear_id = req.params.bear_id
       console.log('bearId: ',bear_id)
       bears.list = bears.list.filter(item => +item.id !== +bear_id)
```

```
        res.json(bears)
    })

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

Checkpoint 1: จงสร้าง API ที่สามารถทำ CRUD กับ JSON ต่อไปนี้ พร้อมตั้งชื่อตัวแปร ฟังก์ชัน URL ให้เหมาะสมกับการทำงาน

REST API - Next.JS Fetch

Frontend API calling with axios

จากตัวอย่าง node ที่ทำ RestAPI (Backend) เพื่อให้ฝั่ง React (Frontend) สามารถเรียกใช้ฟังก์ชันการทำงาน ในการจัดการข้อมูลของ bears ผ่าน CRUD API ต่าง ๆ สามารถเขียน React Hook เพื่อจัดการข้อมูลได้ดังนี้

สร้าง Next.js project

```
$ npx create-next-app bear
$ npm i -s axios
```

/pages/index.js

```
import React, { useState, useEffect } from 'react'
import axios from 'axios'

const URL = `http://localhost/api/bears`

export default () => {
  const [bears, setBears] = useState({})
  const [bear, setBear] = useState('')
  const [name,setName] = useState('')
  const [weight,setWeight] = useState(0)
  const getBears = async () => {
      const result = await axios.get(URL)
      setBears(result.data.list)
  }
  const getBear = async (id) => {
      const result = await axios.get(`${URL}/${id}`)
      console.log('bear id: ', result.data)
      setBear(result.data)
```

```
  }
  const addBear = async (name, weight) => {
      const result = await axios.post(URL,{
          name,
          weight
      })
      console.log(result.data)
      getBears()
  }
  const deleteBear = async (id) => {
      const result = await axios.delete(`${URL}/${id}`)
      console.log(result.data)
      getBears()
  }
  const updateBear = async (id) => {
      const result = await axios.put(`${URL}/${id}`,{
          name,
          weight
      })
      console.log('bear id update: ', result.data)
      getBears()
  }
  const printBears = () => {
      console.log('Bears:', bears)
      if (bears && bears.length)
          return (bears.map((bear, index) =>
              (<li key={index}>
                  {(bear)?bear.name:'-'} : {(bear)?bear.weight:0}
                  <button onClick={() => deleteBear(bear.id)}> Delete </button>
                  <button onClick={() => getBear(bear.id)}>Get</button>
                  <button onClick={() => updateBear(bear.id)}>Update</button>
              </li>)
          ))
      else {
          return (<h2>No bears</h2>)
      }
  }
  useEffect(() => {
      getBears()
  },[])
  return (
      <div>
          <h2>Bears</h2>
          <ul>{printBears()}</ul>

          selected bear: {bear.name} {bear.weight}
          <h2>Add bear</h2>
          Name:<input type="text" onChange={(e)=>setName(e.target.value)} /> <br/>
          Weight:<input type="number" onChange={(e)=>setWeight(e.target.value)} />
<br/>
          <button onClick={ () => addBear(name, weight)}>Add new bear</button>
      </div>
  )
}
```

REST API - SWR

จากตัวอย่าง code ข้างบน หากมีการ fetch data หลายที่ หลาย ๆ ครั้ง แม้ว่าข้อมูลจะซ้ำ หรือไม่ซ้ำกับของเดิม การ fetch API จะต้องเรียก server API ทุกรอบ การแก้ปัญหาที่นิยมใช้คือ นำข้อมูลที่ fetch มาแล้ว เก็บใน Application store เช่น context, redux, recoiled เป็นต้น

- Context - หากข้อมูลมีขนาดใหญ่ และ update บ่อย ๆ จะมีประเด็นเรื่อง ประสิทธิภาพ
- Redux - ใช้งานได้ดี เสถียร แต่ โปรแกรมค่อนข้างซับซ้อน
- Recoil - แก้ปัญหาความซับซ้อนของ Redux แต่ยังออกมาได้ไม่นาน ยังไม่เสถียร

ในส่วนนี้ จึงขอแนะนำ SWR: https://swr.vercel.app/getting-started

SWR มาจาก Stale-While-Revalidate โดยมีหลักการทำงาน ดังนี้

1. เมื่อเรียกข้อมูลผ่าน API ตัว fetcher ก็จะเรียก API ปรกติ และ cache ไว้ (Stale - ข้อมูลเก่า)
2. เมื่อมีการเรียกครั้งต่อไป fetcher จะนำข้อมูลใน cache มาแสดงผลก่อน (ถ้าไม่มีข้อมูลใน cache ก็จะ load ใหม่) ทำให้การแสดงผล ทำได้อย่างรวดเร็ว
3. ระหว่างที่แสดงผลข้อมูลจากข้อมูลใน cache ไปแล้ว fetcher ก็ fetch ข้อมูลมาเปรียบเทียบกับข้อมูลใน cache หากข้อมูลมีการเปลี่ยนแปลง ก็จะ re-render ใหม่อัตโนมัติ (Revalidate)

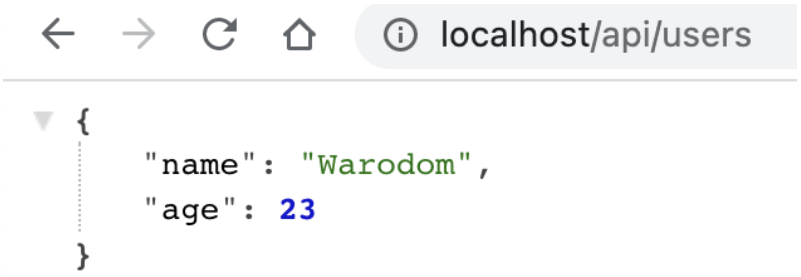
```
$ npm i -s swr
```

Fetch name and mutate

Backend (user.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 user = { 'name': 'Warodom', 'age': 23 }

router.route('/users')
   .get((req, res) => res.json(user))
   .put((req, res) => {
       user = { name: req.body.name, age: user.age }
       res.json(user)
   })

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

Output:



/pages/name-swr.js

```
import useSWR, {mutate} from 'swr'
import axios from 'axios'
const fetcher = url => axios.get(url).then(res => res.data)

const URL = `http://localhost/api/users`

const Name = () => {
   const { data, error } = useSWR(URL, fetcher, { revalidateOnFocus: false })
   if (error) return <div>failed to load</div>
   if (!data) return <div>loading...</div>
   console.log('Home: ', data)

   const updateName = async (name) => {
       const result = await axios.put(URL, {name})
       console.log('Name updated ', result.data)
   }

   return <div>
       Hello:  {data.name}  <br/> Age: {data.age} <br/>
       <button onClick={async () => {
           mutate(URL, {...data, name: "John"}, false)
           await updateName("New John")
           console.log('new data', data)
           mutate(URL,data)
       }}> Mutate </button>
   </div>
}

export default Name
```

ในส่วนของ fetcher **นั้น สามารถใช้** axios **หรือ** fetch **ก็ได้**

```
import axios from 'axios'
const fetcher = url => fetch(url).then(r => r.json())
```

การใช้ mutate

```
mutate(URL, [ ...data ,...newData ])
```

กรณีที่ API **ส่งค่ามาเป็น** pure array (**ไม่ใช่** JSON key value) **จะไม่สามารถใช้การ** mutate **เพราะ** element NewData **จะไปต่อกับ** data **จะไม่ทับกัน (เหมือนในกรณีของ** JSON)

นอกจากนี้ หาก data **กับ** newData **เป็นค่าเดียวกัน** mutate **จะไม่** update cache
หรือ ใส่ argument **ตัวที่** 2 **ไปตัวเดียว ก็จะไม่** update cache

การใช้ swr กับ Bear

/pages/bear.js

```
import React, { useState } from 'react'
import axios from 'axios'
import useSWR, { mutate } from 'swr'

const URL = `http://localhost/api/bears`
const fetcher = url => axios.get(url).then(res => res.data)
const SWR1 = () => {

   // const [bears, setBears] = useState({})
   const [bear, setBear] = useState('')
   const [name, setName] = useState('')
   const [weight, setWeight] = useState(0)

   const { data } = useSWR(URL, fetcher)
   if (!data) return <div>Loading...</div>
   // console.log(data)

   const printBears = (bears) => {
       console.log('Bears:', bears)
       if (bears && bears.length)
           return (bears.map((bear, index) =>
           (<li key={index}>
               {(bear) ? bear.name : '-'} : {(bear) ? bear.weight : 0}
               <button onClick={() => deleteBear(bear.id)}> Delete </button>
               <button onClick={() => getBear(bear.id)}>Get</button>
               <button onClick={() => updateBear(bear.id)}>Update</button>
           </li>)
           ))
       else {
           return (<h2>No bears</h2>)
       }
   }

   const getBear = async (id) => {
       const result = await axios.get(`${URL}/${id}`)
       console.log('bear id: ', result.data)
       setBear(result.data)
```

```
    }

    const addBear = async (name, weight) => {
        const result = await axios.post(URL, { name, weight })
        console.log(result.data)
        mutate(URL)
    }

    const deleteBear = async (id) => {
        const result = await axios.delete(`${URL}/${id}`)
        console.log(result.data)
        mutate(URL)
    }
     const updateBear = async (id) => {
        const result = await axios.put(`${URL}/${id}`,{
            name,
            weight
        })
        console.log('bear id update: ', result.data)
        mutate(URL)
    }

    return (<div>
        <h1> Bear </h1>
        <ul>{printBears(data.list)}</ul>

        selected bear: {bear.name} {bear.weight}
        <h2>Add bear</h2>
           Name:<input type="text" onChange={(e) => setName(e.target.value)} /><br/>
           Weight:<input type="number" onChange={(e) => setWeight(e.target.value)}/>
<br />
        <button onClick={() => addBear(name, weight)}>Add new bear</button>

    </div>)
}

export default SWR1
```

Checkpoint 2: จงสร้าง API ที่สามารถทำ CRUD กับ JSON ของ Checkpoint 1 โดยการใช้ SWR พร้อมตั้งชื่อตัวแปร ฟังก์ชัน URL ให้เหมาะสมกับการทำงาน