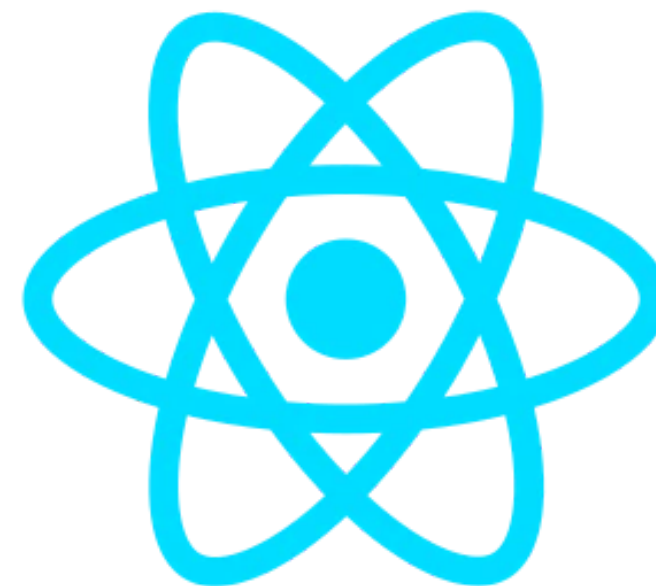


Week 10: 進階 React

許博翔



React JS

作業繳交區

- Flash Card: <https://classroom.github.com/a/ydqcyw4F>
- Calculator: <https://classroom.github.com/a/nQbplxmX>

<FlashCard /> 的 key

上禮拜的寫法：

```
vocabularies.map((v, i) => (  
  <div className="card" key={i}> // 這個 key 很重要！！  
    <Card  
      word={v.word}  
      part_of_speech={v.part_of_speech}  
      definition={v.definition}  
    />  
  </div>  
))
```

加入「查看我的最愛」後，可能會有問題：

- 單字的 index 是可能會改變的

➡ 可以把 key 改成別的 unique 的東西

```
<div className="card" key={v.word}>
```

補充一下上禮拜稍微提過的 Lifting state up

- 常常會出現多個 components 需要存取、修改同一個 state 的狀況
- 把 state 提升到所有有使用到這個 state 的 component 的最近共同祖先 (closest common ancestor) 中

需要先知知道 React 兩個資料處理的理念：

- **Single source of truth**

- 一份資料只應該維護一份
- 能被計算出來的資料不應該另外存成另一份資料

- **Top-down data flow**

- 資料只應該由上而下傳遞
- 一份資料只應在一個地方修改
- 用 `state`、`prop` 和 `event` 實踐

如果下層 component 需要通知上層 component 改動，就透過 event 發送通知

Thinking in React

1. Break the UI into a component hierarchy
2. Build a static version in React
 - top-down v.s. bottom-up
 - 先不需要 state
3. Find the minimal but complete representation of UI state
 - 找出會用到的 state
4. Identify where your state should live
 - lifting state up
5. Add inverse data flow

來做個 Todo List 吧

Pre-work

1. Fork the [repository](#).
2. Clone the forked repo.
3. Run `yarn` or `npm install`.
4. Run `yarn start` or `npm run start`
5. Install [Material UI](#)

```
npm install @mui/material @emotion/react @emotion/styled
```

```
//or
```

```
yarn add @mui/material @emotion/react @emotion/styled
```

Step 1: Break the UI into a component hierarchy

Step 2: Build a static version in React

Step 3: Find the minimal but complete representation of UI state

Step 4: Identify where your state should live

Lifting state up 優缺點

- Pros:
 - 更新資料容易 (Single source of truth)
 - 不會有資料不一致的問題 (Single source of truth)
 - 取用資料時不會不知道應該取用哪份資料 (Single source of truth)
 - 只要往上找就可找到資料源頭 (Top-down data flow)
 - 資料必須在創建的地方修改，減少追查資料更新位置的時間成本 (Top-down data flow)
 - Cons:
 - 每次資料都必須重新計算，某些情況下效能較差 (Single source of truth)
 - 傳遞太多層時會造成維護不易 (Top-down data flow)
- 補充：可以用 `Context` 來解決

Step 5: Add inverse data flow

一個加速開發的小技巧：Conditional rendering

- loading

```
if (loading) {  
  return <h1>Loading...</h1>;  
}  
return <MyComponent data={data}/>
```

- todo list

```
if (isChecked) {  
  return <li className="item">{name} ✓</li>;  
}  
return <li className="item">{name}</li>;
```


不想 render 任何東西 ➡ **return null**

```
if (isChecked) {  
  return null;  
}  
return <li>{name}</li>;
```

更精簡的寫法：Conditional (ternary) operator (? :)

把這個

```
if (isChecked) {  
  return <li>{name} ✓</li>;  
}  
return <li >{name}</li>;
```

寫成這樣

```
return (  
  <li>  
    {isChecked? name + '✓' : name}  
  </li>  
);
```

還有另一種寫法: &&

```
return (  
  <li>  
    {name} {isChecked && '✓'}  
  </li>  
);
```

用網址分成不同頁面：React-Router

Client & Server

- Client（前端）：以網頁來說就是你的瀏覽器、電腦，發送 request 到 Server 端
- Server（後端）：收到 request 開始處理資料，完成後會回傳 response 到 Client 端

安裝

```
$ npm install react-router-dom
```

```
// or
```

```
$ yarn add react-router-dom
```

HashRouter v.s. BrowserRouter

- HashRouter：頁面路徑最前面會有個 "#"，換url時不會發送 request
- BrowserRouter: 頁面路徑不會有井字，但換 url 時會發送 request

```
import { HashRouter } from "react-router-dom";  
import { BrowserRouter } from "react-router-dom";  
  
import { NavLink, Switch, Route } from "react-router-dom";
```

假設現在某個網頁有以下頁面配置：

- `"/home"`：主畫面
- `"/blogs"`：顯示所有部落格文章列表
- `"/contact"`：聯絡資料
- `others`：404 error

每個頁面對應到的 UI (component)

```
// Layout.js
import { Outlet, Link } from "react-router-dom";

export default function Layout() {
  return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Outlet />
    </>
  );
};
```

每個頁面對應到的 UI (component)

```
// Home.js
export default function Home() {
  return <h1>Home</h1>;
};

// Blogs.js
export default function Blogs() {
  return <h1>Blog Articles</h1>;
};

// Contact.js
export default function Contact() {
  return <h1>Contact Me</h1>;
};

// NoPage.js
export default function NoPage() {
  return <h1>404</h1>;
};
```

在 `index.js` 加入 `<Router />`

```
import React from "react";
import ReactDOM from "react-dom/client";
import { HashRouter } from "react-router-dom"; // <=====  
import App from "../containers/App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(  
  <React.StrictMode>  
    <HashRouter> // <=====  
      <App />  
    </HashRouter>  
  </React.StrictMode>  
);
```

在 App 加入 `<Routes />` & `<Route />`

```
import ReactDOM from "react-dom/client";
import { Routes, Route } from "react-router-dom"; // <=====

export default function App() {
  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        <Route index element={<Home />} />
        <Route path="blogs" element={<Blogs />} />
        <Route path="contact" element={<Contact />} />
        <Route path="*" element={<NoPage />} />
      </Route>
    </Routes>
  );
}
```

v6 之前都是用 `<Route path="/" component={Layout}>` 的寫法

URL Parameters

- `<Route path="/:id" component={SecondPage}/>`
- 在 `SecondPage` 讀取 `id` 的方法為: `props.match.params.id`
- 或是使用 hook `useParams()`
 - `let { id } = useParams()`

實作時間~~~~

Axios : 讓 React 打 API 跟後端 (Server) 要資料

HTTP

- *HyperText Transfer Protocol* 超文本傳輸協定
- Client 端和 Server 端之間的網路傳輸協定
- stateless
- 明文傳輸
 - v.s. HTTPS

常見的 HTTP Request Methods

- **GET**：單純請求某個資源
 - E.g. 進入某個網址、讀取訊息
- **POST**：需要執行一些動作
 - E.g. 登入、傳送訊息
- **PUT**：取代掉整個資源
- **DELETE**：刪除某個資源
- **PATCH**：修改部分資源
- **HEAD**：只要 request 的 header，不要 body

HTTP Response

- **1xx** : 稍等
 - **100 Continue** : Server 成功接收、但 Client 還要進行一些處理
- **2xx** : 成功
 - **200 OK** : 成功
 - **204 No Content** : 成功，但沒有回傳的內容 (E.g.: **DELETE**)
- **3xx** : 重新導向
 - **301 Moved Permanently** : 資源被永久移到其他位置，在下一次發出 request 時，瀏覽器會直接到新位置
 - **302 Found (Moved Temporarily)** : 資源暫時被移到其他位置
 - **304 Not Modified** : 資源沒有改變，可以從快取 cache 拿就好

HTTP Response

- **4xx** : Client 端錯誤
 - **400 Bad Request** : 請求語法錯誤、或資源太大等等
 - **401 Unauthorized** : 未認證，可能需要登入或 Token
 - **403 Forbidden** : 沒有權限
 - **404 Not Found** : 找不到資源
- **5xx** : Server 端錯誤
 - **500 Internal Server Error** : 伺服器出錯（搶票時很可能發生）
 - **501 Not Implemented**
 - **502 Bad Gateway** : 伺服器的某個服務沒有正確執行

React 打 API: Axios

- simple promised based HTTP client for the browser and node.js

```
// npm  
npm install axios  
  
// yarn  
yarn add axios
```

Syntax

- ```
axios({
 method: "get", // "post", "put", ...
 baseURL: "https://jsonplaceholder.typicode.com/",
 url: "/posts/1",
 data: {} // for "post", "put", "patch"
})
```

or

- ```
axios.get(URL).then((response) => {})  
  .catch((error) => {});
```
- ```
axios.post(URL, data).then((response) => {})
 .catch((error) => {})
```

# GET

```
import { useState, useEffect } from "react";
import axios from "axios";

const baseURL = "https://jsonplaceholder.typicode.com/posts/1"; // 可以複製到 browser 看一下

export default function App() {
 const [post, setPost] = useState(null);

 useEffect(() => { // 記得打 API 要在 componentDidMount() 裡面打
 axios.get(baseURL).then((response) => {
 console.log(response);
 setPost(response.data); // 用 `response.data` 拿到資料
 }).catch((error) => {
 console.log(error);
 });
 }, []); // <-- after first render

 if (!post) return null; // conditional rendering

 return (
 <div>
 <h1>{post.title}</h1>
 <p>{post.body}</p>
 </div>
);
}
```

# POST

- ```
/* import ... */

export default function App() {
  /* ... */

  function createPost() {
    axios
      .post(baseUrl, {
        title: "Hello World!",
        body: "This is a new post."
      })
      .then((response) => {
        console.log(response);
        setPost(response.data);
      }).catch((error) => {
        console.log(error);
      });
  }

  if (!post) return "No post!";

  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.body}</p>
      <button onClick={createPost}>Create Post</button>
    </div>
  );
}
```

或是利用 `async` / `await`

```
async function getAllPosts() {  
  try {  
    const response = await axios.get("https://jsonplaceholder.typicode.com/posts");  
    console.log(response);  
  } catch (error) => {  
    console.log(error);  
  }  
}
```


Axios Instance

- 不同的 applications 會有相同的 baseURL (但有不同的 routing)

```
const instance = axios.create({ baseURL: "https://jsonplaceholder.typicode.com" });

const getPosts = async (id) => {
  const response = await instance.get(`/posts/${id}`);
  console.log(response);
}


const getAllPosts = async () => {
  const response = await instance.get("/posts");
  console.log(response);
}
```

Axios response schema

```
{  
  data: {},  
  status: 200,  
  statusText: 'OK',  
  headers: {},  
  config: {},  
  request: {}  
}
```

實作時間~~~~

Deploy

- Static server
- AWS Amplify
- Azure
- Firebase
- GitHub Pages 
- Heroku

GitHub Pages

Step 1: Add `homepage` to `package.json`

```
"homepage": "https://<github-username>.github.io/<project-repo>",
```

Create React App (CRA) uses the `homepage` field to determine the root URL in the built HTML file.

Step 2: Install `gh-pages` and add `deploy` to `scripts` in `package.json`

```
// npm  
npm install --save-dev gh-pages  
  
// yarn  
yarn add --dev gh-pages
```

把以下兩行加入 `package.json` 裡的 `scripts`

```
// npm
"predeploy": "npm run build",
"deploy": "gh-pages -d build"

// yarn
"predeploy": "yarn run build",
"deploy": "gh-pages -d build"
```

The `predeploy` script will run automatically before `deploy` is run.

Step 3: Deploy the site

```
// npm  
npm run deploy
```

```
// yarn  
yarn deploy
```


Step 4: Ensure your project's settings use **gh-pages**

- Make sure GitHub Pages option in your GitHub project settings is set to use the gh-pages branch

Most Popular React UI Component Libraries

- MUI (formerly Material-UI)
- Ant Design (AntD)
- React Bootstrap
- ...

Assignment 2 (deadline: 12/01 14:20)

- 請用 React 實作一個 app，滿足下面的要求
 1. 用 react-router 實作四個頁面
 - `/` 首頁：需要有另外三個頁面的 link
 - `/flashcard` 單字卡：上禮拜的 bonus 作業（有 state 的版本）
 - `/translate` 翻譯
 - `/about` 個人資料：需要有姓名、學號、email、照片
 2. (bonus) 有使用到任何 UI Library
 3. (bonus) 切換至 translate 頁面的時候，會自動 focus 在搜尋框
 4. (bonus) 把網頁 deploy 到 Github-Pages

Assignment 2 (deadline: 12/01 14:20)

/translate

- 使用 翻譯 API
- 取得可翻譯的語言列表
- 輸入待翻譯文字
- 選擇要翻譯的語言
- 翻譯並顯示結果

繳交連結：<https://classroom.github.com/a/IS8F00FE>

References

- Ric's Web Programming Class Slides
- <https://reactjs.org/docs/getting-started.html>
- <https://ithelp.ithome.com.tw/articles/10246939>
- <https://create-react-app.dev/docs/deployment/>
- https://www.w3schools.com/react/react_router.asp
- https://yakimhsu.com/project/project_w4_Network_http.html