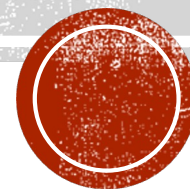


RESTFUL API

林新德

shinder.lin@gmail.com



0. 參考專案

本課程後端參考

<https://bitbucket.org/lsd0125/mfee36-node.git>

本課程前端參考

<https://bitbucket.org/lsd0125/mfee36-next.git>

網站使用 Google 帳號登入 (後端)

<https://github.com/shinder/signin-google-try.git>



1. Restful API

- REST為Representational State Transfer（表現層狀態轉換）的縮寫。
- 2000年由 Dr. Roy Thomas Fielding 在其博士論文中提出的 HTTP 資料交換風格。



1.1 什麼是 Restful API

- 要點：

1. 以 **URI** 指定資源，使用 **HTTP** 或 **HTTPS** 為操作協定。
2. 透過操作資源的表現形式來操作資料。
3. 就是以 **HTTP** 的 **GET, POST, PUT, DELETE** 方法對應到操作資源的 **CRUD**。
4. 資源的表現形式沒有限定，可以是 **HTML, XML, JSON** 或其它格式。
5. **REST** 是設計風格並不是標準，所以沒有硬性的規定。

- 實作 **REST** 的 後端 **API** 一般稱作 **Restful API**



1.2 以商品資料為說明

- 取得列表：
 - `http://my-domain/products` (GET)
- 取得單項商品：
 - `http://my-domain/products/17` (GET)
- 新增商品：
 - `http://my-domain/products` (POST)
- 修改商品：
 - `http://my-domain/products/17` (PUT)
- 刪除商品：
 - `http://my-domain/products/17` (DELETE)



1.3 管理端 URI (需要呈現表單)

- 呈現新增商品的表單：
- `http://my-domain/products/add` (GET)
- 呈現修改商品的表單：
- `http://my-domain/products/17/edit` (GET)
- 呈現刪除商品的表單：
- `http://my-domain/products/17/delete` (GET)



1.4 API 實作

- 商品 API :

- 1. 列表
- 2. 搜尋
- 3. 單項商品

- 購物車 API :

- 1. 讀取
- 2. 加入
- 3. 移除
- 4. 修改
- 5. 清空





2 前端 AJAX

- XMLHttpRequest
- Fetch API
- Axios（套件）
- jQuery.ajax（jQuery 附屬功能）



2.1 XMLHttpRequest

- XMLHttpRequest 屬性及方法參考：<https://developer.mozilla.org/zh-TW/docs/Web/API/XMLHttpRequest>
- 事件：onreadystatechange、onload、onerror、onprogress、onabort。
- 範例：https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest
- 必須在呼叫 `open()` 方法開啟請求連線之前，就註冊好事件監聽器，否則事件監聽器將不會被觸發。
- 可選擇同步或非同步。
- 上傳進度，使用 `XMLHttpRequest.upload` 物件的事件：`load`、`error`、`progress`、`abort`。



ajax-xhr-01.html

```
function doAjax() {  
    var xhr = new XMLHttpRequest();  
  
    xhr.onreadystatechange = function (event) {  
        console.log(xhr.readyState, xhr.status);  
        console.log(xhr.responseText);  
        if(xhr.readyState===4 && xhr.status===200) {  
            info.innerHTML = xhr.responseText;  
        }  
    };  
    // xhr.open('GET', 'data/sales01.json', true); // 非同步  
    xhr.open('GET', 'data/sales01.json', false); // 同步  
    // XMLHttpRequest.open(method, url[, async[, user[, password]])  
    xhr.send();  
}
```



2.2 Fetch API

- 參考 https://developer.mozilla.org/zh-TW/docs/Web/API/Fetch_API/Using_Fetch
- 使用 **Promise** 包裝的 **API** 。
- 只有網路錯誤或其他會中斷 **request** 的情況下，才會發生 **reject** 。
- 缺點：無法取得上傳或下載的進度過程。
- 在傳送少量資料時，是方便的選擇。
- 不會主動傳送 **cookie**，需設定 **credentials** 為 **include** 。
- 放棄任務使用 **AbortController** 的 **signal** 。



fetch: 使用 **POST** 方法

```
const url = 'https://example.com/profile';
const data = {username: 'example'};

fetch(url, {
  method: 'POST',
  body: JSON.stringify(data),
  headers: new Headers({
    'Content-Type': 'application/json'
  })
}).then(res => res.json())
  .catch(error => console.error('Error:', error))
  .then(response => console.log('Success:', response));
```



fetch: 上傳檔案及表單

```
const formData = new FormData(document.myForm);

fetch('https://example.com/profile/avatar', {
  method: 'PUT',
  body: formData
})
  .then(response => response.json())
  .catch(error => console.error('Error:', error))
  .then(response => console.log('Success:', response));
```



2.3 Axios

- Axios 工具: <https://www.npmjs.com/package/axios>

- Axios優點

1. 方便使用，類似 jQuery 的 AJAX 方法
2. Promise API 包裝
3. 可以在後端 Node.js 中使用
4. 體積輕量



axios: 上傳檔案及表單

```
axios.post('/try-upload3', formData, {
  headers: {
    'Content-Type': 'multipart/form-data'
  },
  onUploadProgress: function(progressEvent){
    const perc = Math.round(progressEvent.loaded/progressEvent.total*100);
    const t = new Date().toLocaleString();
    const str = `${perc} % : ${t}`;
    console.log(str);
    progress.innerHTML += str+'<br>';
  }
}).then(r=>{
  console.log(r.data);
  console.log(new Date());
});
```





3. 使用 JSON Web Token

- JSON Web Token 社群官網 <https://jwt.io/>
- 使用 <https://www.npmjs.com/package/jsonwebtoken> 套件。
- 先決條件：資料傳送過程必須在加密的環境中使用，如 HTTPS
- 優點：可在不同的用戶端環境使用，不局限於網站。
- 缺點：需存放在用戶端，由 JavaScript 發送，或其他前端技術發送。
- 過期時間 `exp`，也可以使用套件的設定 `expiresIn`。
- （使用 `bcryptjs` 套件加密用戶密碼）



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  
) ☐ secret base64 encoded
```



3.1 JWT 編碼解碼

```
// 成功登入，將可以辨別用戶的資料加密為 JWT（編碼）  
const token = jwt.sign({  
  sid: r1[0].sid,  
  account: r1[0].account,  
}, process.env.JWT_SECRET);
```

```
const auth = req.get('Authorization');  
res.locals.loginUser = null;  
if(auth && auth.indexOf('Bearer ')===0){  
  const token = auth.slice(7);  
  // JWT 解密  
  res.locals.loginUser = jwt.verify(token, process.env.JWT_SECRET);  
}
```



3.2 登入功能

後端

```
app.post('/login', async (req, res)=>{
  const output = {
    success: false,
    code: 0,
    error: ''
  };
  if(! req.body.email || !req.body.password){
    output.error = '欄位資料不足'
    return res.json(output)
  }

  const sql = "SELECT * FROM members WHERE email=?";
  const [rows] = await db.query(sql, [req.body.email]);
  if(! rows.length) {
    // 帳號是錯的
    output.code = 402;
    output.error = '帳號或密碼錯誤'
    return res.json(output)
  }
}
```



```
const verified = await bcrypt.compare(req.body.password, rows[0].password);
if(!verified){
  // 密碼是錯的
  output.code = 406;
  output.error = '帳號或密碼錯誤'
  return res.json(output)
}
output.success = true;
// 包 jwt 傳給前端
const token = jwt.sign({
  id: rows[0].id,
  email: rows[0].email
}, process.env.JWT_SECRET);
output.data = {
  id: rows[0].id,
  email: rows[0].email,
  nickname: rows[0].nickname,
  token,
}
res.json(output)
});
```



登入功能前端

```
function sendData(e) {
  e.preventDefault();
  fetch("/login", {
    method: "POST",
    body: JSON.stringify({
      email: document.form1.email.value,
      password: document.form1.password.value,
    }),
    headers: {
      "Content-Type": "application/json",
    },
  }).then((r) => r.json()).then((data) => {
    if(data.success){
      const obj = {...data.data}
      localStorage.setItem('auth', JSON.stringify(obj))
      alert('登入成功')
    } else {
      alert(data.error || '帳密錯誤')
    }
  });
}
```



3.3 Token 以 headers 傳送

用戶端 Demo: 已經在 localStorage 存有 token 時

```
const jwtData = JSON.parse(localStorage.getItem('auth'));

function send(){
  fetch('/address-book/verify2',{
    headers: {
      Authorization: 'Bearer ' + jwtData.token
    }
  })
  .then(r=>r.json())
  .then(obj=>{
    console.log(obj);
  })
}
```



伺服器端 ExpressJS Middleware Demo

```
app.use((req, res, next)=>{
  res.locals.sess = req.session;
  let auth = req.get('Authorization');
  if(auth && auth.indexOf('Bearer ')==0){
    auth = auth.slice(7);
    jwt.verify(auth, process.env.TOKEN_SECRET, function(error, payload) {
      if(!error){
        res.locals.jwtData = payload; // 解密後的資料掛在 res.locals 上
      }
      next();
    });
  } else {
    next();
  }
})
```





4. NextJS (pages router) 設定

```
// next.config.js
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: false,

  // 將 server origin 設定為環境變數
  env: {
    API_SERVER: 'http://localhost:3002'
  }
}

module.exports = nextConfig
```



4.1 自訂 URL 設定

```
// src/config/ajax-path.js
export const SERVER = 'http://localhost:3600';

export const AB_GET_LIST = `${SERVER}/address-book/api`;
export const AB_GET_LIST_AUTH = `${SERVER}/address-book/api-auth`;

export const LOGIN_API = `${SERVER}/login-jwt`;
```



4.2 JSX 裡的類似 for 迴圈

```
{  
  Array(11)  
    .fill(1)  
    .map(  
      (v, i) => {}  
    )  
}
```

```
{  
  [...Array(11)].map(  
    (v, i) => {}  
  )  
}
```



4.3 NextJS 路由架構

- Pages router 以 `pages/` 為根目錄。

檔案路徑	對應的 URL
<code>pages/login.js</code>	<code>http://localhost:3001/login</code>
<code>pages/products/index.js</code>	<code>http://localhost:3001/products</code>
<code>pages/products/[pid].js</code>	<code>http://localhost:3001/products/17</code>
<code>pages/slug1/[slug2]/[slug3].js</code>	<code>http://localhost:3001/slug1/aaa/bbb</code>



```
// *** pages/slug1/[slug2]/[slug3].js ***  
import { useRouter } from "next/router";  
  
export default function Slug3() {  
  const router = useRouter();  
  
  return <pre>  
    { JSON.stringify(router.query, null, 4) }  
  </pre>;  
}
```

http://localhost:3001/slug1/aaa/bbb

```
{  
  "slug2": "aaa",  
  "slug3": "bbb"  
}
```





5. 以分頁按鈕取得分頁資料

```
// ***** pages/products/index.js *****  
// 列表頁 1  
export default function Products() {  
  const router = useRouter();  
  const [data, setData] = useState({  
    redirect: "",  
    totalRows: 0,  
    perPage: 4,  
    totalPages: 0,  
    page: 1,  
    rows: [],  
  });  
  const [keyword, setKeyword] = useState("");
```



```
// 列表頁 2
useEffect(() => {
  setKeyword(router.query.keyword || "");
  const usp = new URLSearchParams(router.query);

  fetch(`${process.env.API_SERVER}/products?${usp.toString()}`)
    .then((r) => r.json())
    .then((data) => {
      console.log(data);
      setData(data);
    });
}, [router.query]);
```



```
{/* 列表頁 3 */}
<ul className="pagination">
  {Array(5)
    .fill(1)
    .map((v, i) => {
      const p = data.page - 2 + i; // 目前頁碼取前兩個
      const query = { ...router.query };
      if (p < 1 || p > data.totalPages) return;
      query.page = p;
      return (
        <li className={ `page-item ` + (p === data.page ? "active" : "") }
          key={p}
        >
          <Link
            className="page-link"
            href={"?" + new URLSearchParams(query).toString()}
          > {p} </Link>
        </li>
      );
    })}
</ul>
```



```
<table className="table table-bordered table-striped"> { /* 列表頁 4 */}  
  <thead>  
    <tr>  
      <th>#</th>  
      <th>書名</th>  
      <th>作者</th>  
      <th>價格</th>  
      <th>書號</th>  
    </tr>  
  </thead>  
  <tbody>  
    {data.rows.map((i) => (  
      <tr key={i.sid}>  
        <td> {i.sid} </td>  
        <td>  
          <Link href={"/products/" + i.sid}> {i.bookname} </Link>  
        </td>  
        <td>{i.author}</td>  
        <td>{i.price}</td>  
        <td>{i.book_id}</td>  
      </tr>  
    ) )}  
  </tbody>  
</table>
```



5.1 點擊項目進入單項商品頁

```
<td>  
  <Link href={"/products/" + i.sid}> {i.bookname} </Link>  
</td>
```



```

export default function ProductItem() {
  const router = useRouter();
  const [row, setRow] = useState(null);
  useEffect(() => {
    fetch(process.env.API_SERVER + "/products/" + router.query.pid)
      .then((r) => r.json())
      .then((result) => {
        if (result.success) {
          setRow(result.row);
        } else {
        }
      });
  }, [router.query]);
  return (
    <>
    <Navbar />
    <div>ProductItem</div>
    <p>{router.query.pid}</p>
    <pre>{row ? JSON.stringify(row, null, 4) : <p>沒有資料</p>}</pre>
    </>
  );
}

```





6. 兄弟元件間的資料共享

- ** 複習內容
- 單向資料流的限制。
- 只能透過上層元件，來共享資料或者操作資料的方法，如 `setState()`。
- **CompA** 為父元件，**CampB** 和 **CampC** 為其子元件。




```
// *** components/CompA.js
import {useState} from "react";
import CompB from '@components/CompB'
import CompC from '@components/CompC'

export default function CompA() {
  const [count, setCount] = useState(1);
  console.log('CompA render--')
  return (
    <div style={{border: '2px solid red'}}>
      <div>CompA</div>
      <CompB setCount={setCount} count={count}/>
      <CompC count={count}/>
    </div>
  );
}
```



```
// *** components/CompB.js
export default function CompB({count, setCount}) {
  console.log('CompB render--')
  return (
    <div style={{border: '2px solid green'}}>
      <div>CompB</div>
      <button
        onClick={() => {
          setCount((prevVal) => prevVal + 1);
        }}
      >
        {count}
      </button>
    </div>
  );
}
```



```
// *** components/CompC.js
export default function CompC({count}) {
  console.log("CompC render--");
  return (
    <div style={{border: '2px solid blue'}}>
      <div>CompC</div>
      <h2>count: {count}</h2>
    </div>
  );
}
```





7. 使用 Context (react.js)



7.1 網頁明暗模式 ThemeContext

```
// *** contexts/ThemeContext.js
import { createContext, useState } from "react";

export const themes = {
  dark: {
    name: 'dark',
    backgroundColor: 'gray',
    color: 'white',
  },
  light: {
    name: 'light',
    backgroundColor: 'yellow',
    color: 'black',
  }
};

const ThemeContext = createContext({});
export default ThemeContext;
```



7.2 使用 Context Provider

```
// *** contexts/ThemeContext.js

export const ThemeContextProvider = function ({children}){
  const [theme, setTheme] = useState(themes.light);
  return (
    <ThemeContext.Provider value={{theme, setTheme}}>
      {children}
    </ThemeContext.Provider>
  )
};
```

- ThemeContext 和 ThemeContextProvider 在同一個檔案。



7.3 使用 useContext()

```
// *** components/Navbar.js
import { useContext } from "react";
import ToggleButton from "../ToggleButton";
import ThemeContext, { themes } from "../contexts/ThemeContext";

export default function Navbar() {
  // 對應到 value <ThemeContext.Provider value={{...theme, setTheme}}>
  const { theme, setTheme } = useContext(ThemeContext);
  // 其他內容
}
```




```
{/* 按钮切换 theme */}  
<li className="nav-item">  
  <ToggleButton  
    texts={["暗", "亮"]}   
    statusIndex={theme.name === "dark" ? 0 : 1}  
    handler={(i) => {  
      setTheme(i === 0 ? themes.dark : themes.light);  
    }}  
  />  
</li>
```



7.4 登入狀態 AuthContext

```
// *** contexts/AuthContext.js 1/3

import { createContext, useState } from "react";
import { useNavigate } from 'react-router-dom';

const AuthContext = createContext({});

export default AuthContext;
```



```
// *** contexts/AuthContext.js 2/3
export const AuthContextProvider = function ({children}){
  const navigate = useNavigate();
  const unAuth = {
    authorised: false, // 有沒有登入
    sid: 0,
    account: '',
    token: '',
  };
  let initAuth = {...unAuth};
  // 取得目前狀態，看 localStorage 的資料是否表示已登入
  const str = localStorage.getItem('auth');
  if(str) {
    const localAuth = JSON.parse(str);
    if(localAuth && localAuth.token){
      initAuth = {...localAuth, authorised: true}
    }
  }
  const [myAuth, setMyAuth] = useState(initAuth);
```



```
// *** contexts/AuthContext.js 3/3

const logout = ()=>{
  localStorage.removeItem('auth');
  setMyAuth(unAuth);
  navigate('/login');
};

return (
  <AuthContext.Provider value={{myAuth, setMyAuth, logout}}>
    {children}
  </AuthContext.Provider>
)
};
```



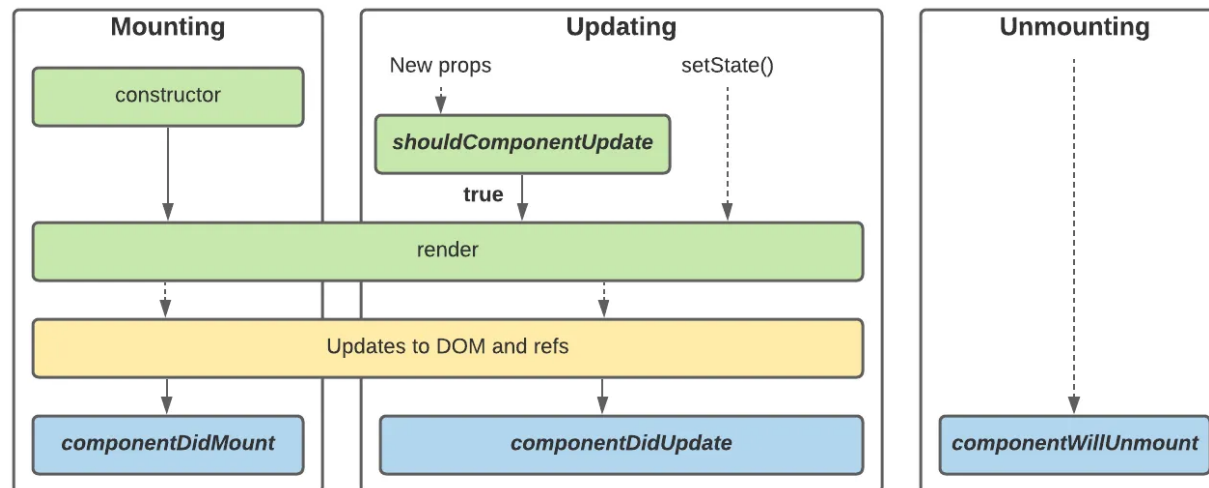
7.5 登入表單

```
// *** pages/Login.js
const mySubmit = async (e) =>{
  e.preventDefault();
  const { data } = await axios.post(LOGIN_API, formData);
  // console.log(data);
  if(data.success){
    localStorage.setItem('auth', JSON.stringify(data.auth) );
    alert('登入成功');
    // console.log(JSON.stringify(data))
    setMyAuth({...data.auth, authorised: true});
    navigate('/');
  } else {
    localStorage.removeItem('auth'); // 移除
    alert('登入失敗');
  }
};
```





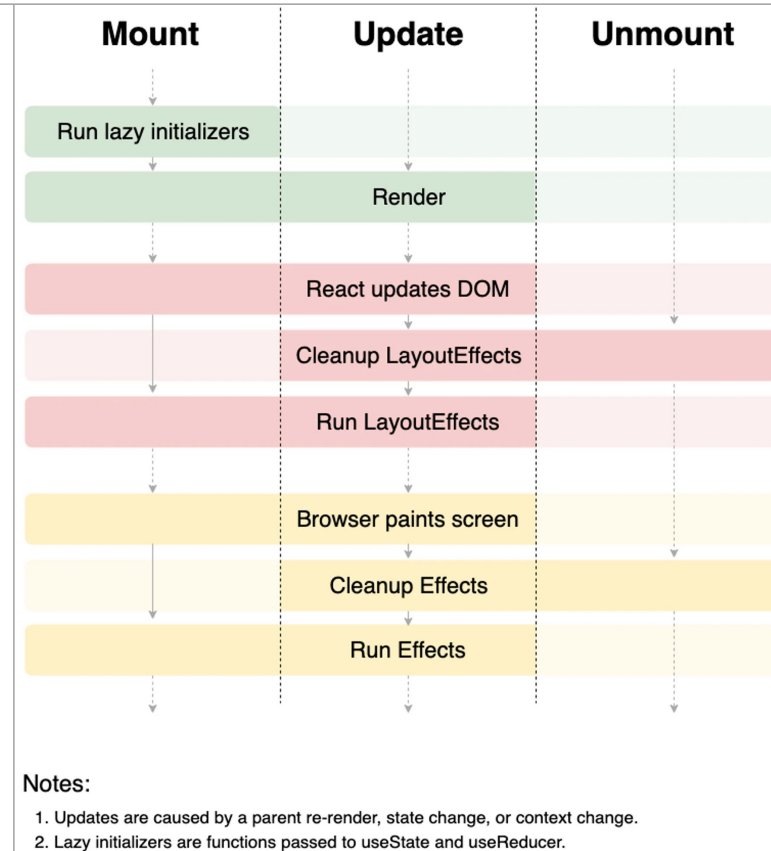
8. 元件生命週期 (參考資料)



React Hook Flow Diagram

v1.3.1 github.com/donavon/hook-flow

<https://bhanuteja.dev/the-lifecycle-of-react-hooks-component>




```
// *** pages/life-cycle.js
import {useState} from "react";
import LifeA from "@components/LifeA";
export default function LifeCyCle() {
  const [show, setShow] = useState(false);
  return (
    <>
      <div className="container">
        <button
          onClick={() => {
            setShow((old) => !old);
          }}
        >
          toggle
        </button>
        {show && <LifeA/>}
      </div>
    </>
  );
}
```



```
// *** components/LifeA.js 1/2
import {useEffect, useState} from "react";

export default function LifeA() {
  const [val, setVal] = useState(10);
  console.log("render----1");
  useEffect(() => {
    console.log("掛載之後");
    let n = 1;
    const interval_id = setInterval(() => {
      console.log(n++);
    }, 1000)
    return () => {
      console.log("將要卸載");
      clearInterval(interval_id)
    }
  }, []);
}
```



```
// *** components/LifeA.js 2/2

useEffect(() => {
  console.log("已更新");
  return () => {
    console.log("将要更新");
  }
}, [val]);
return (
  <div>
    LifeA
    {console.log("render----2")}
  </div>
);
}
```





9. 使用 Canvas (參考資料)

- 若是載入圖檔再繪至 canvas 時，應該在 `useEffect()` 內操作。

<https://bitbucket.org/lsd0125/mfee29-node/src/main/react-try/my-react01/src/pages/Canvas1.js>



9.1 實作快取圖片檔

```
const [cart, setCart] = useState([]);  
const [cache, setCache] = useState({}); // 快取 image 物件
```

```
const getImageFromPath = (path)=>{  
  return new Promise((resolve, reject)=>{  
    if(cache[path]){  
      return resolve(cache[path]); // 回傳已存在的資料  
    }  
    const img = new Image();  
    img.onload = () => {  
      resolve(img);  
      setCache({...cache, [path]:img});  
    };  
    img.src = path;  
  });  
}
```



9.2 重繪時避免閃爍

- 準備兩個 **canvas**，先繪在 **shadow-canvas**，都繪製完成後再繪到 **real-canvas**。

```
<canvas
  ref={shadowRef}
  width="800"
  height="600"
  hidden
></canvas>
<canvas
  ref={realRef}
  width="800"
  height="600"
  style={{ border: "1px dotted blue" }}
></canvas>
```



- 準備兩個 **canvas**，先繪在 **shadow-canvas**，都繪製完成後再繪到 **real-canvas**。

```
const renderCanvas = async ()=>{
  const realCtx = realRef.current.getContext("2d");
  const shadowCtx = shadowRef.current.getContext("2d");
  const bg = await getImageFromPath('/imgs/dish.jpeg');

  shadowCtx.clearRect(0, 0, shadowRef.current.width, shadowRef.current.height);
  shadowCtx.drawImage(bg, 0, 0);

  const tmpCart = cart.slice(0,5);
  for(let i=0; i<tmpCart.length; i++){
    const img = await getImageFromPath(`/imgs/${tmpCart[i].img}`);
    shadowCtx.drawImage(img, i*100, i*100);
  }
  realCtx.drawImage(shadowRef.current, 0, 0);
}
```





10. Google account 登入 (參考資料)

- 先登入 <https://console.cloud.google.com/>
- 1. 到「API 程式庫」>「公開」>「社交」啟用 Google People API。
- 2. 到「API和服務 > 憑證」，點選「+建立憑證」>「OAuth 用戶 ID」。
- 3. 「應用程式類型」選「網頁應用程式」
- 4. 「已授權的 JavaScript 來源」加入正式環境及測試環境的 URI。
- 5. 「已授權的重新導向 URI」加入接收 Query String 參數的頁面 URI，同樣包含測試及正式環境。
- 6. 建立完成後，「下載 OAuth 用戶端」的 JSON 檔。
- 7. 到「OAuth 同意畫面」，設定應用程式名稱，及測試帳號等。



10.1 使用OAuth2Client

```
const { OAuth2Client } = require('google-auth-library');

// 載入 GCP OAuth 2.0 用戶端 ID 憑證
const keys = require(__dirname + '/../client_secret.json');
const oAuth2c = new OAuth2Client(
  keys.web.client_id,
  keys.web.client_secret,
  keys.web.redirect_uris[0] // 測試, http://localhost:3000/callback
);
```



10.2 建立連結 URL

```
router.get('/', async function(req, res, next) {  
  // scopes 參考: https://developers.google.com/people/api/rest/v1/people/get  
  const authorizeUrl = oAuth2c.generateAuthUrl({  
    access_type: 'offline',  
    // 欲取得 email, 要兩個 scopes  
    scope: [  
      'https://www.googleapis.com/auth/userinfo.profile',  
      'https://www.googleapis.com/auth/userinfo.email',  
    ]  
  });  
  
  res.render('index', { title: '點擊連結登入', authorizeUrl });  
});
```



10.3 利用 tokens 取得資料

```
router.get('/callback', async function(req, res, next) {
  const qs = req.query;
  let myData = {};
  if(qs.code){
    // 內容參考 /references/from-code-to-tokens.json
    const r = await oAuth2c.getToken(qs.code);
    oAuth2c.setCredentials(r.tokens);
    const url =
'https://people.googleapis.com/v1/people/me?personFields=names,emailAddresses,photos';

    const response = await oAuth2c.request({url});
    // response 內容參考 /references/people-api-response.json
    myData = response.data;
  }
  res.render('callback', { title: 'Callback result', qs, myData });
});
```



THANK YOU

