# **Part 1：React環境配置**

## **1.1：建立React專案**

① 安裝node.js（<https://nodejs.org/en/download>）

① →**安裝.msi版**

② 檢查node.js有沒有安裝成功

① → 在cmd輸入**node -v**和**npm -v**

③ 打開terminal，輸入以下指令

**npm create vite@latest** \*\*\*如果有問你是不是確定要安裝這個就選yes

A black screen with white text

AI-generated content may be incorrect.

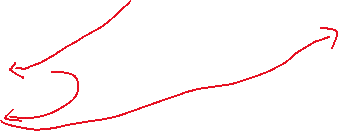
④ 關掉terminal再開過，重新輸入**npm create vite@latest**，然後輸入project名,   
④ framework選擇react，variant選擇javascript

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.



⑤ 看到系統生成文件夾就表示基本設定OK了

A screenshot of a computer

AI-generated content may be incorrect.A yellow folder with a white paper in it

AI-generated content may be incorrect.

⑥ 安裝所有依賴的套件

⑥ I) 進入專案所在目錄，打開terminal → **cd “專案所在目錄”**

⑥ II) 安裝套件 → **npm install**

⑦ 執行server

⑥ **npm run dev**

A screenshot of a cell phone

AI-generated content may be incorrect.

←瀏覽器看到這樣的畫面表示成功了

## **1.2：在電腦打開別人的react專案**

* 我們的react專案中用npm安裝了任何東西，都會記錄到package.json，在不同電腦打開專案後只需要按照package.json記錄的安裝就會把需要的東西都安裝好，然後就可以打開專案了
* **語法：npm install** → 按照package.json安裝需要的檔案

# **Part 2：電腦配置MySQL server**

## **2.1：安裝MySQL Server和MySQL workbench**

* **網址：**
* <https://dev.mysql.com/downloads/installer/>
* <https://dev.mysql.com/downloads/>

\*\*\*在里面兩個都選擇，本地資料庫的密碼設置爲20250101

## A screenshot of a computer AI-generated content may be incorrect.**2.2：導出資料庫的sql檔**

## **2.3：創立新Schema**

# **Part 3：React的前端架構**

## **3.1：React的基本架構介紹**

|  |  |
| --- | --- |
| **JSX** | 在 JS 內寫 HTML的語法 |
| **Components**  **組件 / 元件** | * React把UI中的每一個部分拆解成不同的Components * React只有一個main page，之後每一個頁面都可以視爲一個大Components，里面的每一個部分都是一個小Components * React刷新頁面時是以Components為單位去刷新 |
| **Props** | 父組件傳資料給子元件的機制 |
| **State** | 組件自己的狀態資料（是一種變數，可以隨著更新自動刷新） |
| **Hooks** | 處理組件的state更新 |
| **Router** | 管理多頁面的切換 |

## **3.2：基本架構的範例**

Nav bar

景點列表

景點1

景點2

景點3

景點4

景點資訊

行程列表

* 黑色的是整個頁面
* 綠色的都是小components
* 紫色的也是components，components裡可以包含其他components
* 整個頁面可以視爲一個大components，把所有小components放進大components裡排版就是一整個頁面
* App.js只引入關鍵的components
* 每一次更新時不會刷新整個頁面，以components為單位刷新

## **3.3：React的Router**

* **目的：**因爲React只有一頁，每一次刷新都是以components為單位刷新，所以需要有一個路**目的：**由去引導我們到xx路徑時要刷新出哪一些components。
* **操作的文件：**App.jsx
* **事前準備：**

**① 安裝router （在cmd）**

→ **npm install react-router-dom**

**② 在App.js引入** **（在App.jsx）**

→ **import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';**

* **語法（寫在App.jsx的return裡）：**

|  |
| --- |
| **<Router>**  **<Routes>**  <Route path="**路徑1**" element={**<Components\_1 />**} />  <Route path="**路徑2**" element={**<Components\_2 />**} />  <Route path="**路徑3**" element={**<Components\_3 />**} /> **\*\*\*紫色的可以自己改**  …  **</Routes>**  **</Router>** |

當我們的網址輸入path為路徑1時，頁面就會渲染處Components\_1

* **例子：**

|  |
| --- |
| **<Router>**  **<Routes>**  <Route path="**/**" element={**<HomePage/>**} />  <Route path="**/attraction**" element={**<Attraction\_container/>**} />  **</Routes>**  **</Router>** |

# **Part 4：React前端語法——Components**

## **4.1：Components的概念和寫法**

* **本質：**一種JavaScript 函数（可以使用标签进行扩展），會回傳**一個**元素
* **寫在：**.jsx檔，一個組件就是一個jsx檔**（建議jsx檔和組件名一樣）**
* **寫Components的方式（寫在jsx檔）：**

① **引入React**

**import React from "react";**

② **寫函數組件**

|  |  |
| --- | --- |
| **格式** | const **組件名** = (**參數**) => {  return(  HTML根元素  )  } |
| **例子** | const **UserProfile** = ({ **userName**, **loginTime** }) => {    **return** (  **<div>**  <h2>使用者資訊卡</h2>  <p><strong>使用者名稱：</strong> {**userName**}</p>  <p><strong>登入時間 (UTC)：</strong> {**loginTime**}</p>  **</div>**  );  }; |

**因爲本質上我們寫的是一個javascript的函式，因此javascript可以做的事情都可以在這個函式做，例如把某些變數做運算後再渲染到網頁**

③ **導出組件**

|  |  |
| --- | --- |
| **格式** | export default **組件名**; |
| **例子** | export default **UserProfile**; |

* **注意事項！！！：**

**① 組件的名字要大寫**

**② 必須回傳單一個根元素**

\*\*\*return里只可以有一個元素，如果要回傳多個元素可以用一個空的tag包起來

|  |  |
| --- | --- |
| **錯** | const **UserProfile** = ({ **userName**, **loginTime** }) => {    **return** (  <h2>使用者資訊卡</h2>  <p><strong>使用者名稱：</strong> {**userName**}</p>  <p><strong>登入時間 (UTC)：</strong> {**loginTime**}</p>  );  };  **錯誤：因爲這樣有回傳<h2>和<p>共三個元素** |
| **對** | const **UserProfile** = ({ **userName**, **loginTime** }) => {    **return** (  <>  <h2>使用者資訊卡</h2>  <p><strong>使用者名稱：</strong> {**userName**}</p>  <p><strong>登入時間 (UTC)：</strong> {**loginTime**}</p>  </>  );  };  **正確：回傳的是一大塊元素，里面有多少個小元素不重要** |

## **4.2： Components引入CSS**

**Method 1：Inline CSS**

* **方法：**直接把CSS寫在component函式里寫成一個object，然後在元素的style使用
* **例子：**

|  |
| --- |
| const UserProfile = ({ userName, loginTime }) => {  const **cardStyle** = {  border: '1px solid #ccc',  padding: '20px', **① 把CSS寫成Object**  margin: '20px',  };  return ( **②** **直接把Object寫在style**  <div **style={cardStyle}**>  <h2>使用者資訊卡</h2>  <p><strong>使用者名稱：</strong> {userName}</p>  <p><strong>登入時間 (UTC)：</strong> {loginTime}</p>  </div>  );  }; |

**Method 2：External CSS**

* **方法：**直寫一個單獨的CSS檔，然後在.jsx檔引入（建議css檔名和jsx檔一樣）

**→ import 'css檔';**

* **例子：**

|  |
| --- |
| **UserProfile.css** |
| **cardStyle** {  border: 1px solid #ccc,  padding: 20px,  margin: 20px,  } |
| **UserProfile.jsx** |
| import React from "react";  **import './ UserProfile.css';** //引入UserProfile.css到User Profile.jsx  const UserProfile = ({ userName, loginTime }) => {  return (  <div id="**cardStyle**">  <h2>使用者資訊卡</h2>  <p><strong>使用者名稱：</strong> {userName}</p>  <p><strong>登入時間 (UTC)：</strong> {loginTime}</p>  </div>  );  };  export default UserProfile; |

# **Part 5：React前端語法——Props**

## **5.1：Props的概念**

* **這是什麽：**一種**父組件**傳遞資料給**子組件**的機制
* **本質：**一種變數（由父組件傳遞給子組件）
* **特點：**props 不可變
* **語法：**

① **父組件傳遞**

**<子組件名稱 propName={propValue} />**

**例：**

|  |  |
| --- | --- |
| const Student = () => {  return (  **<Student\_data** **id={1}/>**  )  } | 父組件是Student，它呼喚一個子組件Student\_data，並傳遞一個叫id的變數給它，id的值是1 |

② 子組件接收

**const 子組件 = (props) => {xxx = props.propName}**

**例：**

|  |  |
| --- | --- |
| const **Student\_data** = (**props**) => {  return (  <div>**props.id**</div>  ) | 用一個叫props的object作爲parameter，然後在程式碼里呼喚出它，id是傳遞過來時的props名 |

## **5.2：判斷有沒有接收到props**

* **沒接收到的情況：**如果某個props沒接收到，那召喚它時就會顯示**undefined**
* **語法：{props.參數名 && 返回的元素}**

**語法：**→ 這樣可以對兩個東西做and，如果沒收到參數的話props.參數名就會是**undefined**，  
**語法：**→ 那and的加過就是false就不會返回這個元素

* **例子：**

|  |  |
| --- | --- |
| function Child(**props**) {  return (  <div>  {**props.id** && <p>ID: {**props.id**}</p>}  </div>  );  } | 判斷是否有真的收到props.id，如果沒有的話**<p>ID: {props.id}</p>**也不會顯示  這樣可以避免顯示出  **<p>ID: undefined</p>** |

## **5.3：語法補充——解構**

* **定義：**快速從Object或Array中取出資料，直接賦值給變數
* **語法：**

**① Array解構**

|  |
| --- |
| **const** arr = [10, 20, 30];  **const** [a, b, c] = arr; **// a = 10, b = 20, c = 30** |

**\*\*\*** array的解構一定是**按照順序**依次給a,b,c，

**\*\*\* [a,b,c] = arr** 和 **[b,a,c] = arr**不一樣

**② Object解構**

|  |
| --- |
| **const** person = { name: 'Alice', age: 25 };  **const** {name, age } = person; **// name ="Alice", age = 25** |

**\*\*\*** Object的解構順序不影響

**\*\*\* {name, age} = person**和 **{age, name} = person**不一樣

* **在React的用途：**傳遞props到子元素時

→ 快速解構props，然後直接使用以key命名的變數，不需要一直寫props.xxx

|  |  |
| --- | --- |
| function Child( **{id,name}** ) {  return (  <div>  {**id** && <p>ID: {**id**}</p>}  {**name** && <p>Name: {name }</p>}  </div>  );  } | 直接把傳進來的props解構成**id**和**name**，這樣就可以直接使用**id**和**name**  不用寫props.id和props.name，因爲已經把props解構成id和name兩個變數 |

## **5.4：不同父組件傳遞不同的props**

* 有時候我們可能不同父組件要傳遞不同的props值，React的運作是有傳遞到什麽進來，props這個參數就會有什麽，沒傳遞到就是undefined
* **例子：**

|  |  |
| --- | --- |
| function Child(**props**) {  return (  <div>  {**props.id** && <p>ID: {**props.id**}</p>}  {**props.name** && <p>name: {**props.name**}</p>}  </div>  );  }  **// 父組件1只傳 id**  function Parent1() {  return <Child **id**={123} />;  }  **// 父組件2只傳 name**  function Parent2() {  return <Child **name**="Alice" />;  } | 有哪一個props傳遞到Child中，就顯示那一個，沒有傳遞到的就會是undefined，然後and之後就不會出來 |

# **Part 6：React前端語法——State**

## **6.1：State的概念**

* **本質：**一種

# **Part 7：React前端語法——Hooks**

## **7.1：Hooks的概念**

* **本質：**一種

# **Part 8：React連後端資料庫**

## **8.1：React連資料庫的整體架構**

* **前端** → React框架
* **後端** → node.js中的express
* **原理：在**node.js寫api，在node,js從資料庫獲取資料，然後傳json到react。React則依照自**原原理：**己要的資料傳要求給node.js。

執行sql

MySQL

呼叫Api

Node.js

React

(jsx)

回傳資料

回傳資料(json)

* 寫API
* 監聽API
* 寫前端
* Fetch api

**三個部分都會有自己的server，三個都要打開才能順利讀取到資料**

## **8.2： Node.js的API寫法(在js檔)**

* **Step 1: 先import：express、mysql2和cors**
* **Express的功能：**Node.js 網頁伺服器框架，功能是簡單地寫出 API 或網頁伺服器
* **CORS的功能：**允許不同來源的網頁（不同port的server）來請求這個後端的 API

|  |
| --- |
| const express = require('express');  const mysql = require('mysql2');  const cors = require('cors'); |

* **Step 2: 建立一個 Express 應用並啟用CORS + 設定Port**
* **Port的設定：**後端通常設定爲3001（react前端默認是5173）

|  |
| --- |
| const app = **express()**; **// 建立 Express 應用程式**  **app.use(cors());** **// 讓前端 React 可以存取這個express應用程式**  const **port = 3001**; **// 後端改用 3001 port，避免與前端衝突** |

* **Step 3: 連資料庫**
* **函式：① 連線設定→ const connection = mysql.createConnection(參數看範例)**

**② 正式連線 → connection.connect( )**

* **MySQL密碼：**20250101

**\*\*以上都要儲存在一個變數connection里，之後才能用這個connection去執行sql**

|  |  |
| --- | --- |
| **① 連線設定** | **② 正式連線** |
| const connection = mysql.createConnection({  host: 'localhost',  user: 'root',  password: '20250101',  database: 'students'  }); | **connection.connect(**err => {  if (err) {  console.error('❌ 連線失敗：', err.message);  return;  }  console.log('✅ 成功連線到 MySQL 資料庫！');  }**);** //綠色那段可有可無，用來判斷錯誤的 |

* **Step 4: 開始設定API**
* **函式**

**① 設定API路徑**

**① ✅語法: app.get('path', (req, res) => {收到請求後要執行什麽}  
① ✅例:** app.get('/api/students', (req, res) => { }

**\*\*\*路徑可以包含變數，用「:」表示變數**

**① ✅ 例:** app.get('/api/students/**:id**', (req, res) => { } **// id就是變數，可以用在查詢時**

**① ✅** \*\*\*獲取API路徑中的變數 → **req.params.變數名稱**

**② 執行SQL**

**① ✅語法: connection.query('SQL指令',(err, results) => {要對回傳結果做什麽}**

**① ✅例:** connection.query('SELECT \* FROM students', (err, results) => { }

**①**

**① \*\*\*SQL指令可以包含變數，?表示用，然後引號外用[ ]把變數包起來**

**① ✅例:** connection.query('SELECT \* FROM students WHERE id = ?', [id], (err, results) => { }

**①** \*\*\*以上寫在上一個步驟的{ }内

**③ 顯示出錯誤**

**① ✅語法: res.status(500).json({ error: err.message });**

**①** \*\*\*寫在執行SQL那一段的{ }内

**④ 把查詢到的結果回傳給前端**

**① ✅語法: res.json(results);**

**①** \*\*\*寫在執行SQL那一段的{ }内

* **完整範例**

|  |
| --- |
| app.get('/api/students/:id', (req, res) => {  const studentId = req.params.id; **// 取得 URL 上的 id**  connection.query('SELECT \* FROM students WHERE id=?',[studentId], (err, results) => {  if (err) {  res.status(500).json({ error: err.message });  return; **// 有錯誤就可以直接退出了**  }  res.json(results); // 這裡回傳資料給前端  });  }); |

## **8.3： 在React中用API接收資料庫的資料（寫在jsx）**

**Step 1：useState 宣告**

* **語法：const [state變數, state更新函式] = useState([]);**
* **例子：** const [students, setStudents] = useState([]);

**例子：**// 創立一個students的state變數，然後用setStudents更新

**Step 2：在useEffect呼叫API並更新state中的變數**

* **語法：① 執行useEffect → useEffect(() => {要做什麽}, []);**

**② fetch 請求本地端 API → fetch('API 路徑')**

**③ 把請求到的資料轉成json → .then(res => res.json())**

**④ 把請求到的資料更新到state變數 → .then(data => state更新函式(data))**

* **完整例子：**

|  |
| --- |
| **useEffect(() => {**  **fetch('http://localhost:3001/api/students/1')**  **.then(res => res.json())** //轉成json  **.then(data => setStudents(data))** //用state更新函式更新state變數 PS：data是自己取的名  **.catch(err => console.error('Fetch error:', err));** // 檢測錯誤（可有可無）  **}, []);** |

**Step 3：回傳components到app.jsx**

* **在回傳的過程仍然用state變數去渲染**