

2017-2 Embedded System

Lab1: Chatroom

B02901137 吳元魁 / B02901011 趙祐毅

2017/3/31

1. 分工

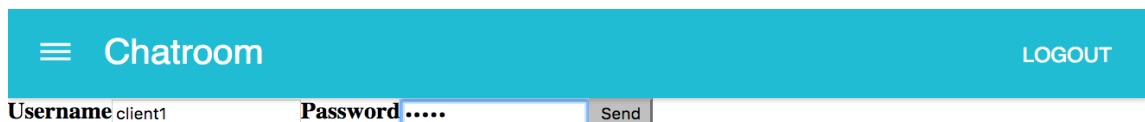
- Server + Database: 趙祐毅
- Client: 吳元魁

2. 程式 Platform

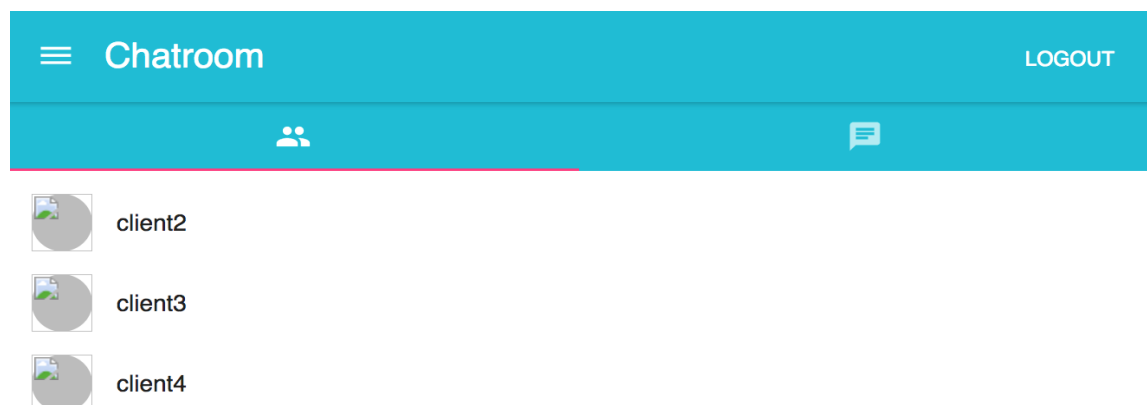
- Server: Node.js “v6.10.1 LTS”
- Database: mySQL
- Client: React, Redux
- Socket: Socket.io
- Module bundler: webpack

3. 系統具備哪些功能

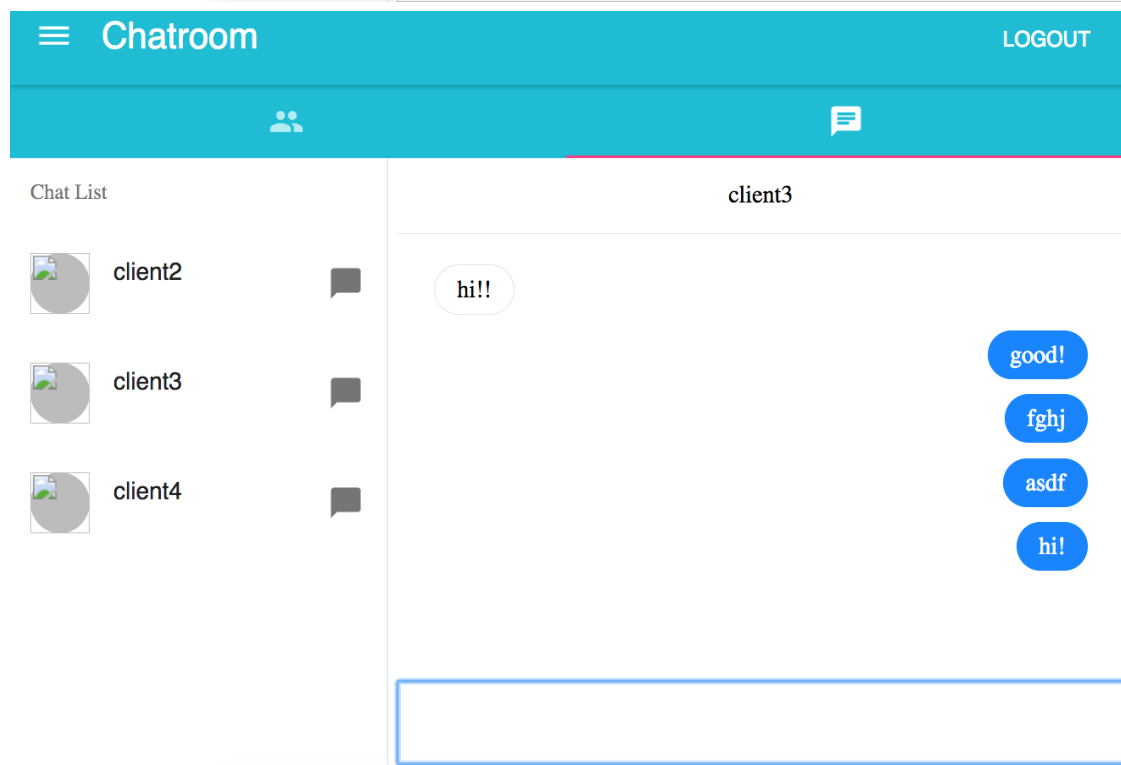
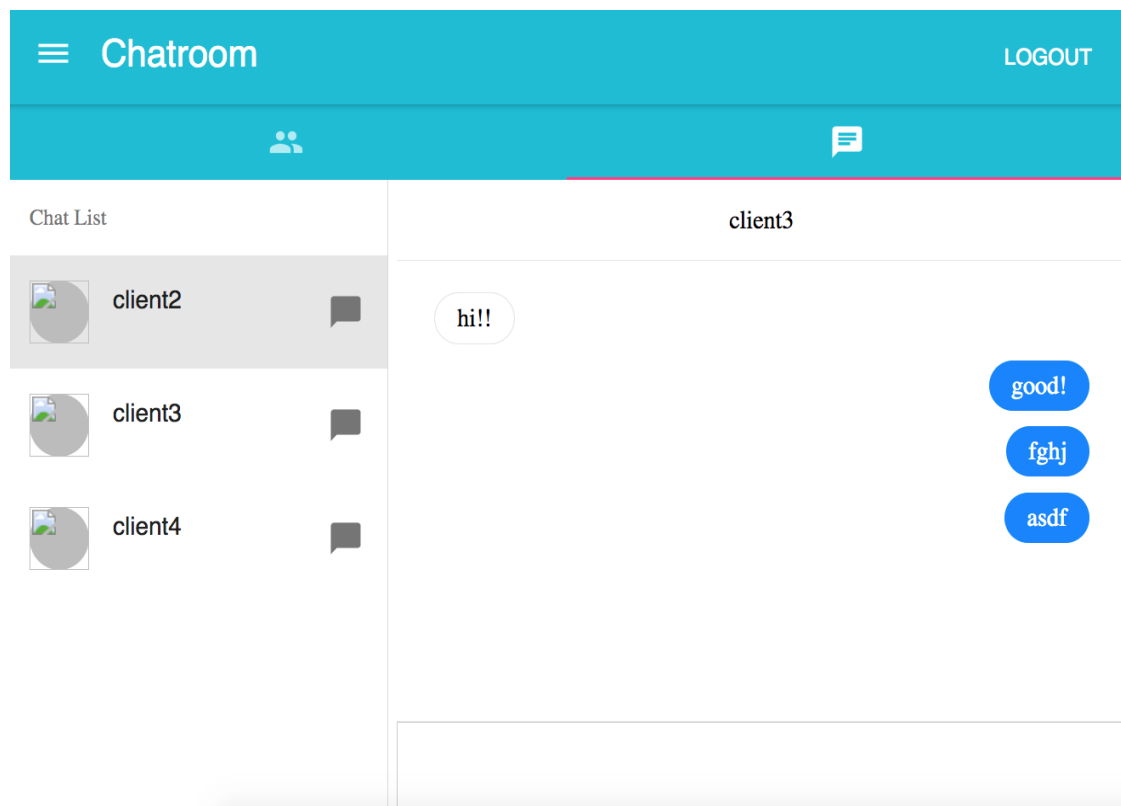
- 登入



- 加好友（未在 Client 端完成）
- 顯示好友列表



- 單獨與不同好友對話&存取好友聊天記錄



- 邊欄

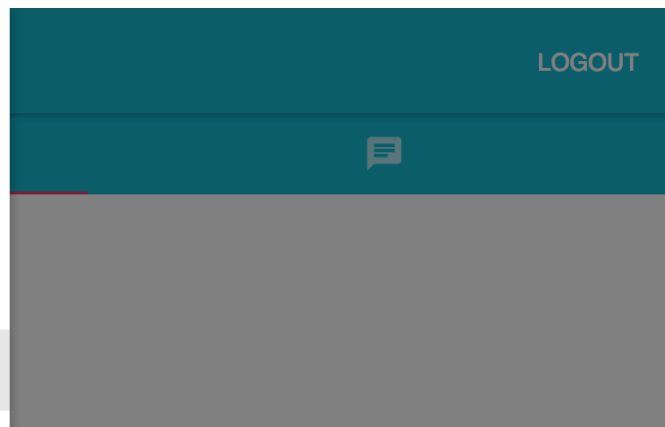
User: client1

Notifications & Sounds

Report a problem

Help

Privacy & Terms



- 登出(未完成)

4. 系統架構

a. 系統包含哪些子元件？

- 後端 Server

要讓基本的 Server 可以運作，主要包含兩個部分：架設 Server、設定 HTML 要求的物件。

- 資料庫 Server

資料庫採用 MySQL，先為系統架設 MySQL server，並建造出一個準備要用的 Database，之後使用 Node.js 中的 Module “Sequalize” 來做管理。可以在 Node.js 中輕鬆的建 table，並且進行存取、查詢的功能。

- 前端 Client

- 登入界面
- 聊天界面
 - 聯絡人名單
 - 對話列表
 - 聊天對話框

b. 每個元件擔任什麼角色？

- 後端 Server

以下對於後端 Server 的三個主要部分做詳細介紹。

- 架設 Server

運用 'http', 'express', 'socket.io' 三個 Module。

```
var express      = require('express');
var app          = express();
var server       = require('http').createServer(app);
var io = require('socket.io').listen(server);
server.listen(3000);
```

以上實作，把 'express', 'http' 套件打開，並且設定出 Server，這個 Server 用 socket.io 來與 client 端做溝通，最後，設定聽的 Port 為 3000。

- 設定 HTML 要求的物件

```
app.use(express.static('public'));
app.get('/', function(req, res){
  res.sendFile( __dirname + "/index.html" )
});
```

前面已經設定好 app = express()，上面第一行設定 HTML 可以存取靜態資料夾，也就是把 ./public 這個資料夾設定為可以存取。

在使用者輸入 ip:3000/ 時，會把目前資料夾底下，index.html 這個 file 丟出去！

● 資料庫 Server

資料庫採用 MySQL，先為系統架設 MySQL server，並建造出一個準備要用的 Database，之後使用 Node.js 中的 Module “Sequelize” 來做管理，“Sequelize” 是 ORM (Object Relational Mapping)，可以幫助不同資料型態的轉換。在 Node.js 中輕鬆的建 table，並且進行存取、查詢的功能。

- Sequelize 使用

❖ 與資料庫 Server 連線

```
var sequelize = new Sequelize('database name', 'username', 'password', {
  host: 'localhost',
  dialect: 'mysql',
  port: 3306,
  timezone: '+08:00',
});
```

❖ 建立 Table

```
var User = sequelize.define('Users', {
  username: Sequelize.STRING,
});
```

從 sequelize 建立 table 相當容易，也不需要去煩惱 MySQL 複雜的資料型態。

String 的長度可以變動，預設是長度是 255。這對於存名字，或是存訊息是個好選擇。

❖ Insert 資料進入 Table

User

```
.build({ username: data.name })  
.save()  
.then(...);
```

只要指定 Table 名稱，用 build (裡面用 js object 樣式)，後續用 .save() 把它存進去，也可以直接加 .then 往下執行，這裡做到非同步 function 的效果。

❖ 從 Table 找資料

```
User.findAll({ where: { username: data.name }  
}).then(/* */);
```

可以設定不一樣的條件找資料，

findAll → 找所有符合條件的

findOne → 找第一個符合條件的

findById → 在 Sequelize 會依序存 ID，可以從 ID 去找資料。

- 資料庫設計

我們運用三個 Table 來存資料：Users, Chat_histories, Friend_lists

❖ Users: 存取所有使用者的基本資料

ID	系統自動存的遞增數列
username	使用者名稱
password	使用者密碼
login_t	上一次登入時間
logout_t	上一次登出時間

❖ Chat_histories: 存每個使用者之間的對話紀錄

ID	系統自動存的遞增數列
fromName	丟訊息的使用者名稱
fromId	丟訊息的使用者 ID
toName	收訊息的使用者名稱
toId	收訊息的使用者 ID
msg	訊息內容

CreatedAt	系統自動存時間（時區一開始定義）， 記錄訊息傳的時間
-----------	-------------------------------

❖ Friend_lists: 記錄使用者之間成為朋友的關係

ID	系統自動存的遞增數列
fromName	加朋友的使用者名稱
fromId	加朋友的使用者 ID
toName	被加朋友的使用者名稱
toId	被加朋友的使用者 ID
CreatedAt	系統自動存時間（時區一開始定義）， 記錄訊息傳的時間

● 前端 Client

- App：包含兩個部份，
 - toolbar，左邊有滑動的功能，顯示 Notification, Report Problem 這些功能。
 - 如果還未 login 的時候，會顯示 login 的界面，登入以後會有 sidebar，可以進入聯絡人和交談列表、對話框兩個頁面。
- Login: 一個可以輸入帳號密碼的 form。
- ChatAppBar：包含左邊的 Dropout 和右邊的登出 button。
- NavBar：有兩個 slide 的 slide bar，包含 ContactList 和 ChatList, ChatBox.
- ContactList：好友的列表，包含頭像和名字。
- ChatList：按 last message 的時間排列對話列表。
- ChatBox：聊天室，對方的對話放置在左邊、自己的放置在右邊，聊天按 Enter 即可送出

c. 元件與元件之間的界面與互動？

● Client 與 Server 的互動

- 登入介面

連上 Server 端之後，需要輸入使用者名稱跟密碼，這兩個資訊會被 Socket.io 以 object 的方式傳到後端，Server 檢視這組使用者名稱和密碼有沒有在 User table 裡面，若有，而且密碼正確，則回傳 { success: true, username: username }。

username 是每個使用者作為聊天室 Socket 送訊息的識別。(但是這樣可能會很容易偽裝，若要做加密則用複雜的 Token 當作使用者的識別。

- 要求使用者的好友名單

在前端需要顯示使用者有哪些好友，因此前端會跟後端 server 要資料，前端送出使用者的 Username 和 ID，server 進 friend_list 裡面找尋之後，回傳好友的清單。

- 與好友們的聊天記錄 (列出最近的一條)

希望像 FB messenger 一樣，在左側顯示上次與哪些朋友對話，因此，前端會向後端要聊天記錄的資料，但對每個朋友只要求一條 (顯示出來)。這時候可以用 Sequelize 強大的搜尋功能，如下：

```
{ limit: 1,  
  order: [ [ 'createdAt', 'DESC' ] ] }
```

這兩行限制了只要找一條，而且順序是按照 CreatedAt (被加入的時間) 做遞減排序。再把找到的資料整理成一個 List，回傳給前端，以顯示出來。

- 要求所有歷史紀錄

當使用者點選特定使用者的時候，需要去找尋這兩個人之間所有的對話紀錄，這個時候，也是運用 Sequelize 的 function，很容易就完成。取到資料之後，再對時間做 Sort，回傳給前端。

- Server 與 mySQL 的互動

Server 與 mySQL 之間，運用 Sequelize 這個強大的 ORM，來完成建置 Table，以及存取資料的媒介。只要在一開始寫入要存取的 Database 的類型(mySQL) 以及 Username, password, database name。就可以進行管理。

d. 使用的 Framework

- 後端

- http, express：實做架設 Server 的部分
- sequelize：Server 與 mySQL 的溝通橋樑
- socket.io：由 socket.on 和 socket.emit 和前端進行 data 的傳送。

- 前端

- React：實做 view 的部份，建立一個一個的 component render 在 root 上
- redux：操控前端的 state，由 store 儲存所有的 state，在利用 dispatch(action)去執行 action，再由 reducer 去更改 state。
- redux-saga：asynctrounous 的 middleware，操控整個流程的 flow。

- `socket.io`：由 `socket.on` 和 `socket.emit` 和後端進行 data 的傳送。
- `webpack, babel`：由 `babel` 去 compile es6，並利用 `webpack` 進行 loader 和 compress。

5. 遇到的問題

● JavaScript 同步、非同步的問題

之前相當熟悉靜態語言如 C++, Python 等語法，這些靜態語言是非同步的，也就是可以一行一行執行程式。接觸到 JS 的時候，雖然知道是完全不一樣的思維，也就是動態語言，有同步執行緒的特性，所有東西可以看成是 Parallel 執行。

因此，在實作上，想要達成 Call 某一個 Function，並且讓這個 Function 回傳一個值。不能用原本在 c++ 的方法，這樣可能產生令人意外的結果，用 `Console.log` 也會印出一些不如人意的東西。`Node.js` 有 `Callback function`, `Promise` 或是 `async/await` 的解法，概念上雖然可以理解，但在實際運用上仍然遇到不少問題，況且 `async/await` 的方法，必須搭配像是 `Babel` 的 `Compiler`，增加很多麻煩。主要原本想做的是用 `Username` 進去 `Users` 的 table 查詢 ID，所以會用 `sequelize` 做 `find`，期望把值回傳回來，後來因為實作上不好解決，而且 `sequelize` 的部分都是用 `.then` 的這種樣式，增加複雜性，後來想辦法繞過這個問題，把 ID 也一併存起來，減少一直做 `find` 的工作。

● node.js 與 mySQL 溝通

在與 mySQL 溝通的部分，原本參考網站上的範例，用 `module mySQL` 中的 `"mysql.createConnection"` 處理，然而，並沒有很好用，必須要用 `"connection.query()"` 的方式把東西傳入 mySQL 中，這樣不只實作上 Code 很醜，也可能出現使用者亂輸入一堆東西，讓我丟進 mySQL 中，把資料庫弄的亂七八糟。

● 前後端溝通的問題

因為是前後端分開來寫，況且，前端並不是用傳統的 HTML 的方式寫，因此在測試的時候，並不知道怎麼去做，只能互相討論彼此要傳什麼東西，直到最後接起來才能正式測試，這樣的開發過程，頗痛苦的，因為彼此都不知道有什麼東西，只能憑著自己的想像去測試。

後來，想到可以直接從 Client 端的 `Console` 窗口做測試，把 server 端打開 `socket.on`，client 端用 `socket.emit` 去 trigger 這個 function，再看看 server 端怎麼反應，client 端也可以再打開 `socket.on` 的通道，看一看接收的資料好是否正確。

● Node.js Debug 的問題

原本在寫 Node.js 的時候，大多使用 Console.log 的方式測試 debug，後來有用 node.js debugger，希望能夠像 C++ 一樣，中斷程式做測試，但是因為 Node.js 處理的是 server 端，不能中斷 console.log 東西，後來也了解 console.log 不見得能夠得到我們想要的結果，因為值可能會在我們不注意的時間變動，所以 console.log 的結果不見得會保證真正跑的時候會 work。

另外，我們也用 Nodemon 這個 Module，可以在程式更動的時候自動重新開始跑，不用自己關掉重開，增加編寫的效率。

- 存取 ID 的考慮

一開始在考慮要不要給 Username 一個 unique 的 ID，或是只要 Username unique 就好，後來決定要，雖然會增加一些麻煩，但是可以更好的辨認使用者是誰，或是用 ID 來開 Socket 的 Room。

- 資訊安全的問題

我們再傳訊息的時候，希望前端給後端自己的 Username，以及要傳給誰。但是如果前端隨便傳一個 Username，是不是就可以偽裝自己成為別人，並且亂傳訊息？因此，我們要求前端傳入自己的 ID 作為識別。

但是，覺得聊天室還有很多關於資安的問題需要去考量，才不會動不動就出問題，讓駭客輕易入侵。

- Facebook 登入的測試

我們原本希望做到用 Facebook 登入，因此參考 Tutorial，學習怎麼用 Facebook 的 API，後來有成功跑起來。Facebook 先讓開發者在網路上註冊程式，後來會給一組認證碼，可以嵌入程式中，在使用時，會讓使用者輸入帳密，但登入後會傳登入成功的訊息，並且有一組識別的 ID，讓開發者不會直接觸碰到使用者的帳密，增加資料安全性。

6. 結論與心得

這次的聊天室實作，讓我們實際去觸碰前後端的運作，並且認識 JavaScript 語法的特性、長處、短處。實際遇到很多 Bug，參考很多網路上的資訊，才拼拼湊湊出一些可以 work 的東西。相當有趣，也學習到一塊不同的領域。

JS 是一個廣泛使用的語言，且 Web 也是具有相當的功能，可以更有彈性，在不同的 Platform run。在這次機會，不只學習如何架 Server，也能夠讓 Server 與 Client 溝通，並且學習 Client 端的開發，相當過癮。

在開發的時候，有考慮很多問題，也不斷嘗試與修正。以下幾個是可以之後進一步去研究的方向：

- 希望更進一步了解要開發一個大的聊天室，怎麼加強資安的問題。
- 讓使用者可以登入跟註冊
- 讓使用者可以傳輸文字之外的東西（如：圖片、音訊等）

最後，因為網頁是個快速開發的語言，所以可能兩三年內就有重大變動，這對於我來說會有些 Confuse，不知道學哪種才是最保值的語言，不過我想各語言的概念都不太一樣，在後端也許用 JavaScript 是會持續運用與發展，前端則百家爭鳴，相當有趣！
