ESLab 2017 Spring Lab 1 Report

組員：B02901032 張桓誠 B02901068 林楷恩

Web chat application

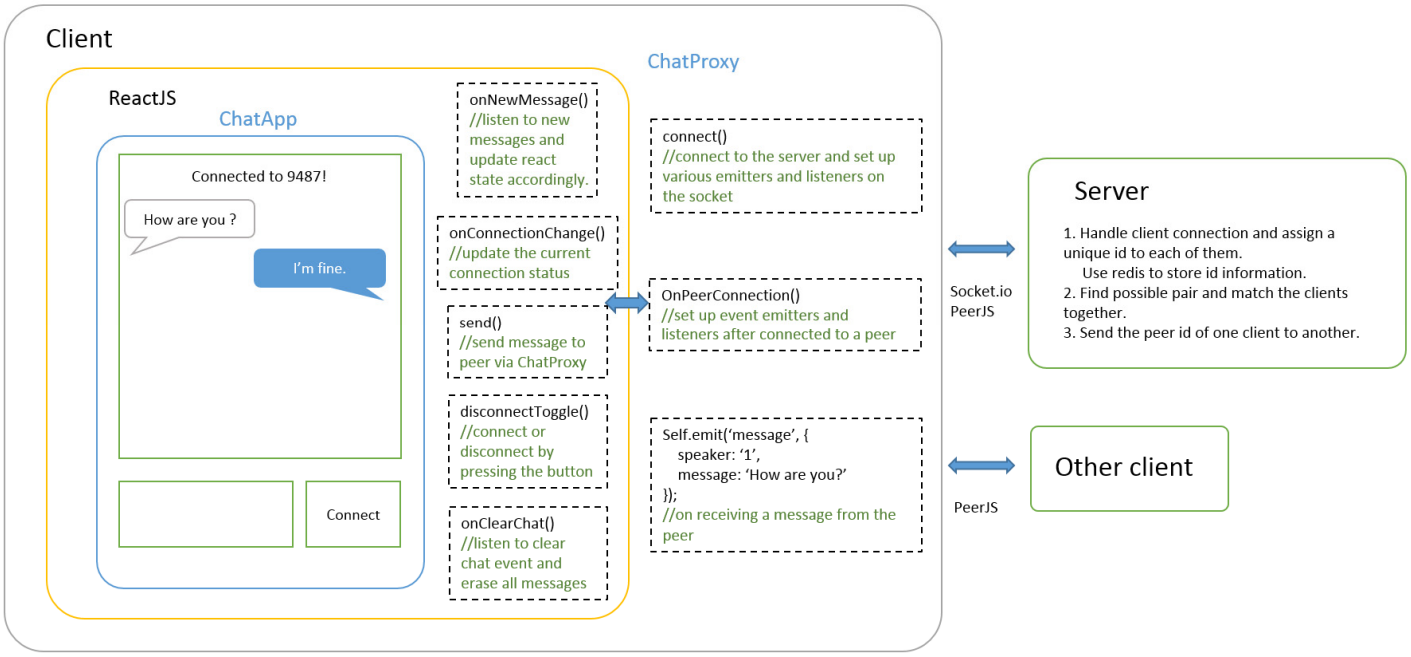
功能特色：

1. P2P 隨機配對
   在一開始構想時，我們打算仿造 WooTalk 做出一個隨機配對的聊天室軟體，利用 Server 隨機分配使用者的配對，然後再利用 P2P 的方式使兩名使用者之間可以直接對話。
2. 即時聊天
3. 匿名對話
   由於記錄使用者資訊較為敏感，且會需要一個完整的資料庫和登入系統，會因此轉移了開發時的資源，我們便不打算記錄使用者的資訊，而是採取匿名制，隨登隨聊。

使用套件：

1. ReactJS
   負責產生靜態 UI，最大的好處是能夠把 UI 以不同 component 的方式呈現，較容易 debug 以及模組化開發。
2. PeerJS
   為了達成 P2P 所使用的套件。
3. Restify
   負責建立 API endpoint，雖然這次的 project 只有拿來 serve static files，但以後若有需要可以再開新的 endpoint。
4. Webpack
   此套件主要作用在於方便管理其他套件，同時在開發的過程可以建立一個即時更新的 server，加速開發過程。
5. Redis
   In-mem DB，感覺會比 MariaDB 快一點點所以決定採用，主要存放連線中用戶的狀態。這次 Project 只是用來儲存狀態而已，沒有用於其他操作，未來有更多功能時可以使用。
6. Socket.io
   傳送聊天對象的 ID 給 client。由於 ID 是由 Server 配對後傳送過去，配對前可能有較長的等待時間，因此決定不讓 client 送 HttpRequest 到 API endpoint 索取 ID，改由 socket 通知 server 狀態改變，配對完成後再由 socket 送回。

架構概述：



我們的做法將整個 app 分為伺服器端和使用者端，利用 Socket.io 和 PeerJS 進行使用者—伺服器，和使用者—使用者之間的訊息傳遞。而如同上圖所示，使用者和外界互動的主要控制元件是 ChatProxy，它使用到了 Node.js 內建的 Event Emitter，這樣的做法可以達成非同步的事件觸發，並且比較不會干擾到 React 內部的 life cycle。當非同步的事件觸發 callback 之後，UI 就會收到訊息並且更新至最新狀態，如此一來，我們便可以看到新的訊息出現在螢幕上。至於當要發送訊息給對方時，也會透過 ChatProxy 的 function 去做處理：當使用者按下 enter 時，便會呼叫 send()這個函數，然後將訊息藉由 PeerJS 傳出去。

架構細節：

伺服器端：

- PeerServer
  - event - connection：新的使用者上線，給予該用戶自己的 peer id
  - event - disconnect：使用者離線，回收 peer id 並將該使用者的資料從資料庫移除
- Socket
  - event - connection：新的使用者上線，給予 socket id
  - event - disconnect：使用者離線，回收 socket id
  - event - agent: client：送訊息過來，代表使用者狀態改變。分別處理新使用者或新聊天對象請求，會更新使用者的狀態資料以及將他們送進等候名單裡。
- (Anonymous) Pairing
  每兩秒執行一次，將等候名單裡的使用者兩兩配成一對，並將 peer id 由 socket 送至 client，使其連線並開始對話。因為使用者離線時不會被從等待名單裡移除，配對時會先檢查雙方是否都仍在線上，若有任一方不在則剔除，持續遞補。

使用者端：

- React UI

  這部分之中最主要的元件是 ChatApp，作用是由上到下的方式去建構 UI，方便控制各自的邏輯以及 CSS 樣式：

  - ChatApp：包住整個網頁的介面，並且記錄使用者的對話等等狀態，並且包含我們會使用到的各種函式，包含了監聽事件觸發的 onNewMessage()、onConnectionChange()、onClearChat()、發送訊息 send()、按下 Connect 按鈕後的 disconnectToggle()等等。另外像是比較簡單的元件如按鈕，也是在 ChatApp 內部定義的。
  - MessageItem 系列：我們把對話用條列式的方式呈現，因此 UI 上便利用三種 component 去達成這個功能，這三種的不同就只有在 CSS 呈現上會對應到使用者自己、對方、系統，分別利用不同的設計使得使用起來更舒適。
  - SaveButton：方便使用者下載對話紀錄的功能，將屬於該功能的函式分開寫入，較為整齊乾淨。

- ChatProxy

  前面有提過這個部分主要的作用是負責和 server 及 peer 之間溝通，因此我們放了許多 event emitter 以及 listener 來達成非同步事件的觸發。會選擇這個方法主要的原因是 React 在每次 render 不同物件的時候，都會有一個 life cycle，因此若讓 React 和伺服器直接互動，會發生狀態沒有確實記錄的問題，就是因為在 life cycle 之中錯誤的地方去存取 UI 的狀態，才導致沒有記錄，而有資料消失的問題。不過比較正規的做法應該是利用 React 搭配的 Redux 來達成控制物件狀態，但是由於 Redux 較為複雜，對於這種簡單的 chat app 是殺雞用牛刀，因此就不選擇使用他。

  ChatProxy 繼承了 EventEmitter 這個物件，利用 emit 和 addListener()，我們就可以自由的設定事件監聽以及觸發的條件，只有在觸發 callback 時才和 UI 互動，所以之間的溝通比較不會太過複雜，也符合 life cycle 的模式。以下將介紹幾個用到的 function:

  - StatusChanged：這個 function 負責和 server 通知 client 現在處於哪個狀態，我們分為 INIT 以及 NEWREQ，一開始初始化的狀態便是 INIT，而後當 client 要找尋 peer 的時候就會改成 NEWREQ，這樣 server 才會將他排入分配的 queue 當中。
  - connect：這個 function 主要的作用在於初始化 socket 及 peer 的事件監聽，並且規定了收到訊息後處理的規則。
  - OnPeerConnection：處理和另外一個 peer 的訊息以及事件，像是對方發送訊息過來，或者是離線等等。
  - onNewMessage、onClearChat、onConnectionChange：這些主要的用途都是讓 UI 能夠知道 ChatProxy 觸發了什麼樣的事件，然後再用 callback 的方式更新 UI。