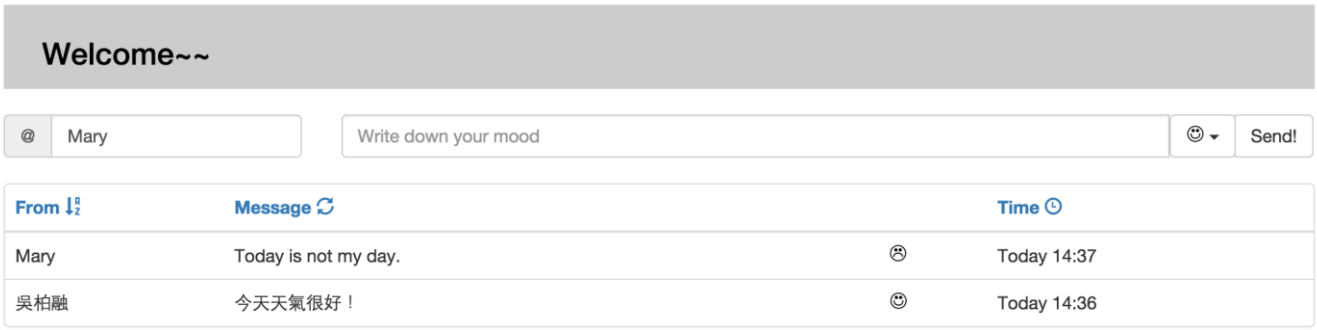
Embedded System Lab HW1 Report

Team 9

欒浩偉 B99202039

吳柏融 B01901119

- **What our board looks like :**



- **Functions provided by our board :**
  1. **A clean and simple UI**.
  2. Four basic functions as mentioned in the homework requirements
  3. Each data contains the **sender's name**, **the message** , **mood information** and **time**.
  4. The server only returns the newest 15 data to client.
  5. User can sort by the **sender's name, message**or **time**
  6. The user can choose the mood of a collection of images we provide before sending the message.

- **How to use the board :**

- What problems we encountered and how we solved:

1. **HTML GET Issue:**

   When I was writing the server side, I found the client side couldn't load CSS and Javascript files. It only showed HTML file and the bootstrap library. And the console kept showing error : 'uncaught SyntaxError: Unexpected token < ' . After searching for solution, I found that it is because the server didn't respond the CSS and javascript file. So I add a condition in **getRequstHandler function** to get the request for CSS , javascript or image file and add a new **function handleStaticPage** to respond to the client. Thus solved the problem.

2. **KISS Issue:**

   I tried to make the code follows KISS simple, and I found that it works well by leveraging the **object-orient property** of javascript. To demonstrate it, **Fig 1**. is my code overview in client.js. All the main functions are stored in *Board.message*, with other helper functions as in literacy. As you can see, it is highly understandable and easy to maintain.

3. **Remembering Issue:**

   I'd like to make the script remember the sorting method previously specified. For example, after I have assigned "sort_by_sender" as my sorting method, the next refreshing will

remember it and sort the refreshed messages by sender name. To implement it, my first solution is **creating a hidden HTML input** as

```
<input type="hidden" id="sortMethod" name="sortMethod"
value="sort_time">
```

,

and furthermore, writing some scripts to get/change its value. It works well by storing such information on DOM, but I later came up a somewhat better solution: leveraging the javascript's **closure** feature! I declare a variable named *sortMethod* in the object *Board*. And since it's maintained by the closures of its "member functions", I can get-change this value at will anytime without dealing with DOM, which usually lacks of efficiency.

```
$(document).ready(function() {
    var Board = {
        init : function() {
            this.setupVar();
            this.setupUi();
            this.bindEvents();
            this.message.retrieve();
        },
        helper : { ... },
        message : {
            prependNew : .. ,
            send : .. ,
            post : .. ,
            retrieve : ..
        },
        setupVar : function() { .. },
        setupUi : function() { .. },
        bindEvents : function() { .. },
    };
    Board.init();
});
```

Fig 1. KISS principle in client.js

4. **Icon displayed by CSS**

I found that in many large websites, every interrelated small icons are saved within a single picture. After some researches, I got that two **CSS properties**: background-size and background-position can fulfil this demand. By experiment, with m*n icons saved in a single image, background-size needed to be set to max(m, n) * 100%, and background-position needed to be set to (i * 1/(m-1), j * 1/(n-1)) respectively.

Fig 2. All icons saved in single image

5. **Double parsing issue:**

In the beginning, the data retrieved from server cannot work locally. By displaying the response in browser's console, I discovered that each packet object sent to server **has been stringify twice**. To demonstrate it, consider a user clicks the send button. The generated packet is first stringified and sent to the server. After the server get it, it push this stringified_packet to the dataset. When "retrieve" is asked from client, the server picks up an array containing such stringified_packet and stringify it such that it can be sent to the client. Hence, the client needs to parse it, to get an array object; and for each item in this array, **parse it again** to get the packet object. This resolves the issue.