**Week 1:**

Reflections:

This week covers a lot of content related to HTML, CSS and JavaScript, which form the basis of the website. The main differences are:

• HTML structure of the website

• CSS changes its appearance or "style" (thus cascading style sheets)

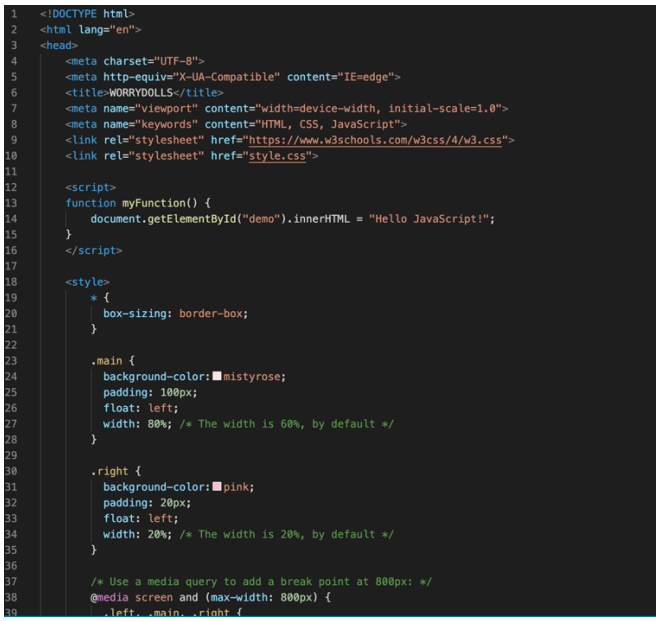
• JavaScript enhances the interactivity of the website, allowing it to adjust according to user information or input.

HTML – divides a web site into multiple elements, such as paragraphs, different title types, links to images, references to other web pages, creating tables, and so on. Html is used to create web pages. It is the markup language of the network. It represents hypertext markup language. HTML describes the structure of a web page and how the browser should display its content.

CSS – set Web page styles, such as background color, font, table and spacing, border, font size and style. CSS represents a cascading style sheet that can be used to decorate the appearance of HTML elements. CSS is a useful tool that can completely change the appearance of web pages.

JavaScript – JavaScript allows websites to be "responsive", thus forming a "responsive web application". Many JavaScript frameworks have also been developed. The goal is to allow developers to access other people's pre - written code and libraries without having to write all the code and block their own files

Task1. Task2 :



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>WORRYDOLLS</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="keywords" content="HTML, CSS, JavaScript">
    <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
    <link rel="stylesheet" href="style.css">

    <script>
    function myFunction() {
        document.getElementById("demo").innerHTML = "Hello JavaScript!";
    }
    </script>

    <style>
        * {
          box-sizing: border-box;
        }

        .main {
          background-color: mistyrose;
          padding: 100px;
          float: left;
          width: 80%; /* The width is 60%, by default */
        }

        .right {
          background-color: pink;
          padding: 20px;
          float: left;
          width: 20%; /* The width is 20%, by default */
        }

        /* Use a media query to add a break point at 800px: */
        @media screen and (max-width: 800px) {
          .left, .main, .right {
```

```
        /* Use a media query to add a break point at 800px: */
        @media screen and (max-width: 800px) {
          .left, .main, .right {
            width: 100%; /* The width is 100%, when the viewport is 800px or smaller */
          }
        }
    </style>

    <style>
    body {background-color:aliceblue;}
    h1 {color:deeppink;}
    h1 {font-size:4vw;}
    h2 {color: hotpink;}
    h2 {font-size:2vw;}
    p1 {color:hotpink;}
    </style>

    <h1 style="font-family:verdana;">WorryDolls</h1>
    <hr>
<head>

<body>
    <!-- Navigation (Stays on Top) -->
    <div class="top bar lightpink">
        <button onclick="document.getElementById('id01').style.display='block'" class="log-button log-green log-large">Login</button>
        <form>
            <label for="username">Username:</label><br>
            <input type="text" id="username" name="username"><br>
            <label for="pwd">Password:</label><br>
            <input type="password" id="pwd" name="pwd">
            <label for="email">Enter your email:</label>
            <input type="email" id="email" name="email">
            <input type="submit" value="Submit">
        </form>

        <button type="button"
        onclick="document.getElementById('demo').innerHTML = Date()">
        Time.</button>
        <p id="demo"></p>
```

Task3:

```
<!DOCTYPE html>
<html>
    <head>
        <Title>Task 3</Title>
        <script src="task3.js"></script>
    </head>
    <body>
        <H1>This is my Responsive JS Page</H1>
        <p id="externalToChange">This element will have more content added to it on load... </p>
        <p id="htmlToChange">I can change the HTML content Via a click.</p>
        <button type="button"
        onclick="document.getElementById('htmlToChange').innerHTML = 'its so simple'">Click
        Me!</button>
        <br>
        <h2>This content is being loaded from my js file</h2>
        <p>Students -- Has Taken Test? -- Test Score</p>
        <table>
            <tbody id="tbody"></tbody>
        </table>
        <h2 id = 'finalOutput'></h2>
    </body>
</html>
```

```
        <button type="button"
        onclick="document.getElementById('demo').innerHTML = Date()">
        Time.</button>
        <p id="demo"></p>
```

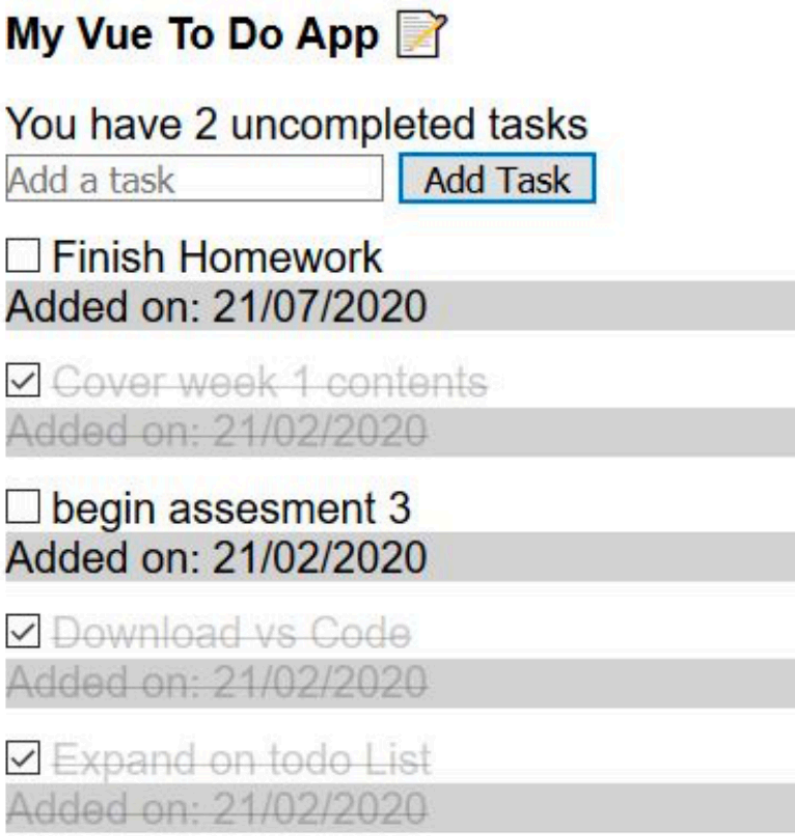
Task4:

```
<!DOCTYPE html>
<html>
<head>
    <title>My first Vue app</title>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script src="task4.js"></script>
</head>
<body>
    <div id="app">
        <h1>My Vue To Do App 📝</h1>
        <b><h4 v-show="remaining > 0">You have {{ remaining }} uncompleted task<span v-show="remaining > 1">s</span></h4>
        <h3 v-show="remaining < 1">Completed all tasks, well done! ✔  </h3>
        </b>
        <input type="text" placeholder="Add a task" v-model="newTodo" autofocus />
        <button class="ui teal right labeled icon button" @click.prevent="addTask">
            <i class="plus icon"></i> Add Task
        </button>
        <ol>
            <li v-for="todo in todos">
                <label>
                    <input type="checkbox" v-on:change="myToggle(todo)" v-bind:checked="todo.done">
                    <del v-if="todo.done">
                        {{ todo.text }} <p id= "date"> Added on:  {{todo.date}} </p>
                    </del>
                    <span v-else>
                        {{ todo.text  }} <p id= "date"> Added on:  {{todo.date}} </p>
                    </span>
                </label>
            </li>
        </ol>
    </div>
</body>
</html>
```

```
new Vue({
    el: "#app",
    data: {
      todos: [
        { text: "Cover week 1 contents", done: true, date: "21/02/2020"},
        { text: "begin assesment 3", done: false , date: "21/02/2020"},
        { text: "Download vs Code", done: true , date: "21/02/2020"},
        { text: "Expand on todo List", done: true , date: "21/02/2020"}
      ]
    },
   computed: {
      remaining: function () {
        var uncompleted = 0;
        for (var index in this.todos) {
          if(!this.todos[index].done) {
            uncompleted++;
```

```
          }
        }
        return uncompleted;
      },
    },

    methods: {
        myToggle: function(todo){
          todo.done = !todo.done
      },
       addT: function () {

       // Add formatted timestamp to each todo
        var today = new Date();
        var dd = String(today.getDate()).padStart(2, '0');
        var mm = String(today.getMonth() + 1).padStart(2, '0');
        var yyyy = today.getFullYear();

              today = dd + '/' + mm + '/' + yyyy;

        if (this.newTodo !== '') {
```

```
          // Add new object to beggining of array
          this.todos.unshift({
            text: this.newTodo,
            done: false,
            date: today
          });

          this.newTodo = '';
        }
      }
    }
  })
```




**Week2:**

Reflections:

This week we learned what responsive webpage design is. In short, responsive design is a method of designing a website. By adjusting the way information is displayed to users, the website looks good on all devices. This can be achieved by considering factors such as device, direction, input, screen size, etc. Using this information, we can make our website respond, so that the information can be effectively displayed in the current environment.

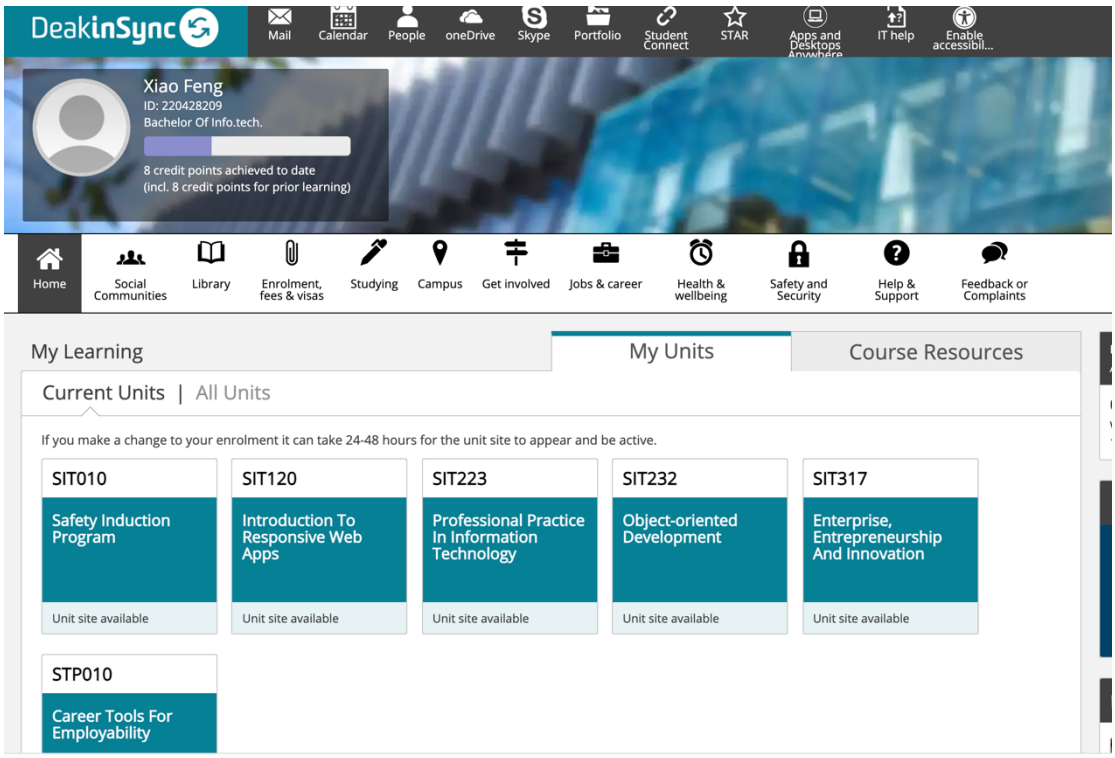
We also learned user stories. Creating a user story is usually the first step in designing a product, because it ensures that we take users into account in our design. From the perspective of users, by creating a short story to explain the specific needs of users for the product, we can focus on delivering a product that implements the functions outlined in each user story.

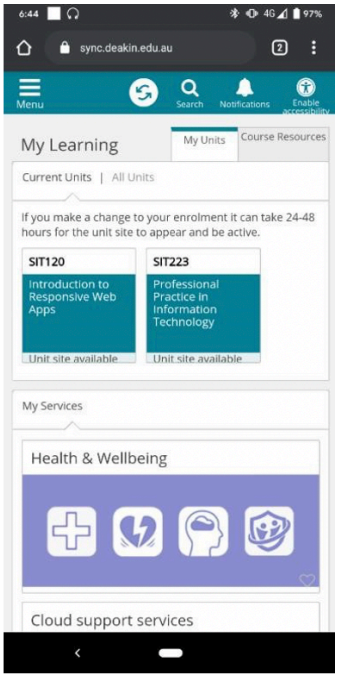
In addition to user stories, we also have acceptance criteria, which is usually a clear list of testable parameters that must be met before considering the completion of user stories. these

This week we also talked about UX (user experience) and UI (user interface). UI is the visual aspect of design. It is what people actually see, click and read, while UX is a little vague because it is more related to the feeling and visibility of the website and is based on the experience of the whole user journey

UI: user interface this is anything a user may interact with. It is the visual aspect of product interface design
UX: This is the user's feeling and perception when interacting with the UI

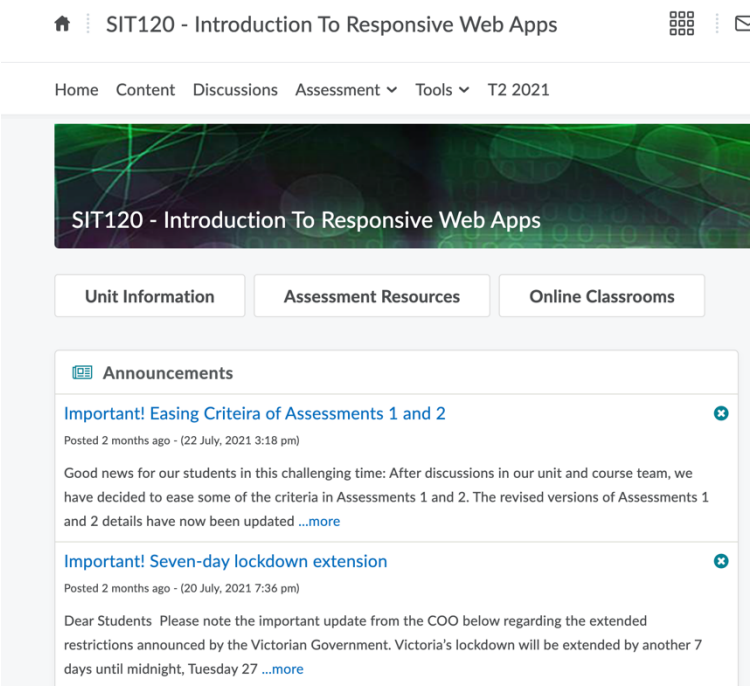
**Task1:**
**Site 1 - deakinsync**.
This site is highly responsive - here are screenshots comparing desktop and mobile versions. Using viewport metatag clearly means that the site is well expanded and easy to use on mobile devices. The menu folds down into a hamburger menu, the weather part is removed, and the help, collaboration and cloud news areas in the second column on the right of the desktop version

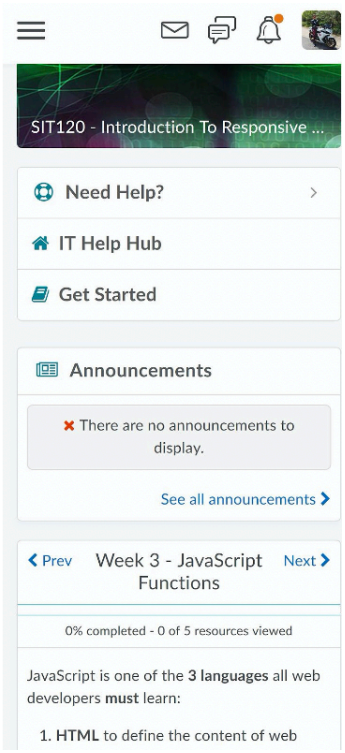





**Site-2 Deakin sit120 unit site**

The unit website is very sensitive. In desktop view, the site has a complete navigation bar at the top, which also has a complete unit name. There are also two main columns on the desktop. The column on the right has support information. In the computer view, we can see that the structure of the layout is basically the same, but the view is reduced, although the font is still readable. In the mobile view, we can see that the navigation is hidden behind the hamburger menu, and the main layout of the page now has only one column, and the most useful information is displayed from top to bottom.






## Task2:

```
<style>
    * {
      box-sizing: border-box;
    }

    .main {
      background-color: mistyrose;
      padding: 100px;
      float: left;
      width: 80%; /* The width is 60%, by default */
    }

    .right {
      background-color: pink;
      padding: 20px;
      float: left;
      width: 20%; /* The width is 20%, by default */
    }

    /* Use a media query to add a break point at 800px: */
    @media screen and (max-width: 800px) {
      .left, .main, .right {
        width: 100%; /* The width is 100%, when the viewport is 800px or smaller */
      }
    }
</style>
```

```
    <div class="main">
    <div class="container center">
        <h2>Main WorryDoll</h2>
        <p1>I am the WorryDoll</p1>
        <hr>
        <img src="doll1.png" class="hover-opacity" style="width:200px">
    </div>
    </div>

    <div class="right">
    <div class="container center">
        <h2>Another WorryDoll</h2>
        <hr>
        <img src="doll2.png" class="w3-hover-opacity" style="width:150px">
        <img src="doll3.png" class="w3-hover-opacity" style="width:150px">
        <img src="doll4.png" class="w3-hover-opacity" style="width:150px">
    </div>
    </div>
```

```
<!-- menu  -->
<div class="menu">
    <!-- menu Title  -->
    <h2 class="menuTitle">WorryDolls</h2>
    <!-- Hamburger menu button with javascript method showMenu() -->
    <a href="javascript:void(0);" v-on:click="showMenu()" class="icon" onclick="myFunction()">
        <i class="fa fa-bars"></i>
    </a>
    <!-- items in menu nav -->
    <div id="menuItems">
        <li><i class="fas fa-search"></i><input class="search" type="text" placeholder="Search"></a></li>
        <li><i class="fas fa-home"></i><a href="index.html">Home</a></li>
        <li><i class="fas fa-book-open"></i><a href="worrys.html">My Worry</a></li>
        <li><i class="fas fa-chart-line"></i><a href="resolved.html">Resolved Worry</a></li>
        <li><i class="fas fa-address-book"></i><a href="share.html">Share to Friend</a></li>
    </div>
</div>
```

```
/* Variables */
:root {
  --primaryTextColor: deeppink;
  --backgroundColor: thistle;
  --SecondaryColor: lightpink;
}
/* reset browser default styles */
* {
  box-sizing: border-box;
  padding: 0;
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
  text-decoration: none;
  list-style: none;
  font-weight: 100;
  /* transition: all 1s ease; */
}
```

```
/* each block corresponds to each rounded square in the ui */
.block {
  padding: 1.5rem;
  border-radius: 15px;
  background: mistyrose;
  transition: all 1s ease;
}
/* submit button */
button {
  float: right;
  width: 5rem;
  height: 2rem;

  background: pink;
  border: none;
  position: relative;
  bottom: 0px;
  right: 0px;
  border-radius: 4px;
}
button:hover {
  background-color: var(--SecondaryColor);
}
button:active {
  background-color: lightpink;
}
```

```css
/* ////////////////////////// Navigation Menu Styling ////////////////////*/
.menu {
  border-radius: 15px 50px 30px 10px;
  grid-area: menu;
  background-color: var(--SecondaryColor);
  padding: 2rem;
  min-width: 240px;
}
.menuTitle {
  font-size: 35px;
}
li {
  padding-top: 2rem;
}
i {
  padding-right: 1em;
}
.search {
  border: none;
  width: 8rem;
  height: 1.5rem;
  border-radius: 10px;
}
#menuItems {
  display: block;
}
/* Remove mobile hamburger icon*/
.icon {
  display: none;
}
```

```css
/* worry input text box */
.worryInput {
  width: 100%;
  height: 80%;
  resize: none;
  font-size: 1.2rem;
  border: none;
  background: pink;
}
/* worry slider */
.slidecontainer{
  width: 100%;
}

.slider{
  width: 40%;
}
/* slider icons */
.fa-frown, .fa-smile{
  font-size: 20px;
  margin: 10px;
  padding: 0;
}

/* welcome block */
.main {
  grid-area: main;

}
/* responsive main title text */
.mainTitle {
  font-size: clamp(30px, 4vw, 80px);
  margin-bottom: 0.6rem;
}
```

**Task 3:**

| User story | Acceptance Criteria |
|---|---|
| As an online trouble recorder, I hope to write down my worry | • The user must be able to enter text in the text box.<br>• The user must be able to confirm after writing input. |
| As an online trouble recorder, I hope to preserve my emotional state. | • Prompt for login or new registration at site launch.<br>• Login should take users to their home dashboard.<br>• Registration should guide the user through the registration process.<br>• The user enters the name, email, and password.<br>• Stored credentials.<br>• Confirmation email has been sent.<br>• Once a new entry is written, users should confirm that they have finished editing it.<br>• After editing, the portal site should have online storage and save it to online storage. |
| I want to automatically add other information, such as date and time, to my Worry entry | • Once the entry is edited, the site must give each entry a time stamp.<br>• You should use JS to generate timestamps to get the device time.<br>• The site should display this timestamp on all entries. |
| I want to remove the worry entry feature. | • There is a delete button on each entry.<br>• Prompts the user to decide whether to delete.<br>• Remove entries from pages and cloud storage. |
| I want to be able to search my entries. | • The site should have a search function.<br>• The search should return specific terms related to the search term. |

| | |
|---|---|
| **I'd like to greet them with my name.** | • **Users must be logged in.**<br>• **The site takes the user name and displays it in HTML format.** |
| **I'd like a menu on the side of the desktop.** | • **Web sites must know screen sizes.**<br>• **Adjust the layout to place the menu on one side of the desktop screen size.** |

**Task 4:**

```
        // Function to get location data and display coordinates to user
        showPosition: function() {
          if(navigator.geolocation) {
              navigator.geolocation.getCurrentPosition(function(pos) {
                  var posInfo = "Latitude: " + pos.coords.latitude + ","+"\n" +
"Longitude: " + pos.coords.longitude;
                  document.getElementById("result").innerHTML = posInfo;
              });
          } else {
              alert("Your browser does not support HTML5 geolocation, Sorry");
          }
      }
```

**Week3:**

Reflections:

This week we learned JavaScript, a network interaction language. Although its popularity began with the front-end client, it is now used on the back-end. Html is the content of the website and CSS is the style, so JavaScript is what we use to make the website interactive and dynamic.

Task1:

```
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
text.length;     // Will return 26
```

Task2:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Methods</h2>

<p>The toString() method converts a number to a string.</p>

<p id="demo"></p>

<script>
let x = 123;
document.getElementById("demo").innerHTML =
  x.toString() + "<br>" +
   (123).toString() + "<br>" +
   (100 + 23).toString();
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>toString()</h2>
<p>The toString() method returns an array as a comma separated string:</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
</script>

</body>
</html>
```

Task3:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript getTime()</h2>
<p>The internal clock in JavaScript counts from midnight January 1, 1970.
</p>
<p>The getTime() function returns the number of milliseconds since then:</p>

<p id="demo"></p>

<script>
const d = new Date();
document.getElementById("demo").innerHTML = d.getTime();
</script>

</body>
</html>
```

Task4:

Computed Properties and Watchers:

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})
```

Class and Style Bindings:

```
<div class="static active"></div>
```

Conditional Rendering:

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

List Rendering:

```
<ul id="example-1">
  <li v-for="item in items" :key="item.message">
    {{ item.message }}
  </li>
</ul>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

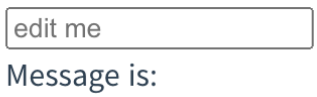
Event Handling:

```
<div id="example-1">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button above has been clicked {{ counter }} times.</p>
</div>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    counter: 0
  }
})
```

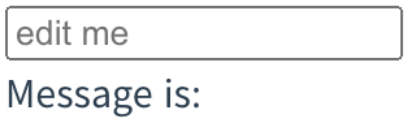
Form Input Bindings:

```
<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```

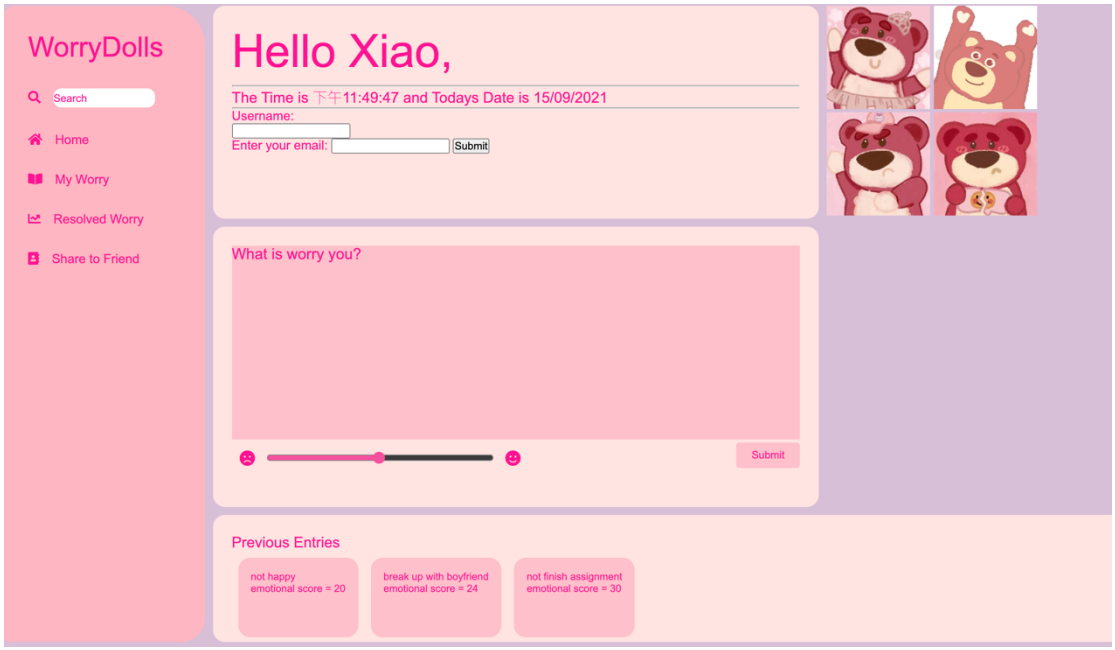



Components Basics:

```
<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```




Project progress:




**Week 4:**

Reflections:

This week we discussed what a framework is and how to use components in Vue. In front-end development, we use HTML, CSS and JS. JS has many variants, which appear in the form of libraries and frameworks. The components in Vue allow us to create self-contained code blocks that can be reused throughout the site. Improve performance and enable us to create large-scale applications. We can think of components as a tree.

Task1:

```
var app = new Vue({
    el: "#app",
    data: {
        time: null,
        // name shown on dashboard
        Name: "Xiao",
        // temporary store for current entry
        message: null,
        // temporary strage for the slider value
        slider: 50,
        // location check
        locate: 0,

        // where saved entries go
        entries: [
            {
                text: 'not happy',
                slider: 20
            }, {
                text: 'break up with boyfriend',
                slider: 24
            }, {
                text: 'not finish assignment',
                slider: 30
            }
        ]
    },
```

Task2:

```
        <div class="worry block">
            <!-- worry text input -->
            <textarea v-model="message" class="worryInput" placeholder="What is worry you?"contenteditable="false"></textarea>
            <!-- worry emotional slider -->
            <div class="slidecontainer">
                <i class="fas fa-frown"></i>
                <input type="range" v-model="slider" min="1" max="100" value="50" class="slider" id="myRange">
                <i class="fas fa-smile"></i>
                <button v-on:click="submit">Submit</button>
            </div>
        </div>
        <!-- previous enteries block -->
        <div class="previous block">
            <h3>Previous Entries</h3>
            <div class="previousEntriesContainer">
                <!-- will iterate through all entries in worry and render them -->
                <div v-for="entry in entries">
                    <p class="previousEntry">{{ entry.text }} <br> emotional score = {{entry.slider}}</p>
                </div>
            </div>
        </div>
        <div class="right">
            <div class="container center">
                <img src="doll1.png"vclass="hover-opacity" style="width:130px">
                <img src="doll2.png" class="hover-opacity" style="width:130px">
                <img src="doll3.png" class="hover-opacity" style="width:130px">
                <img src="doll4.png" class="hover-opacity" style="width:130px">
            </div>
        </div>
    </div>
</div>
```

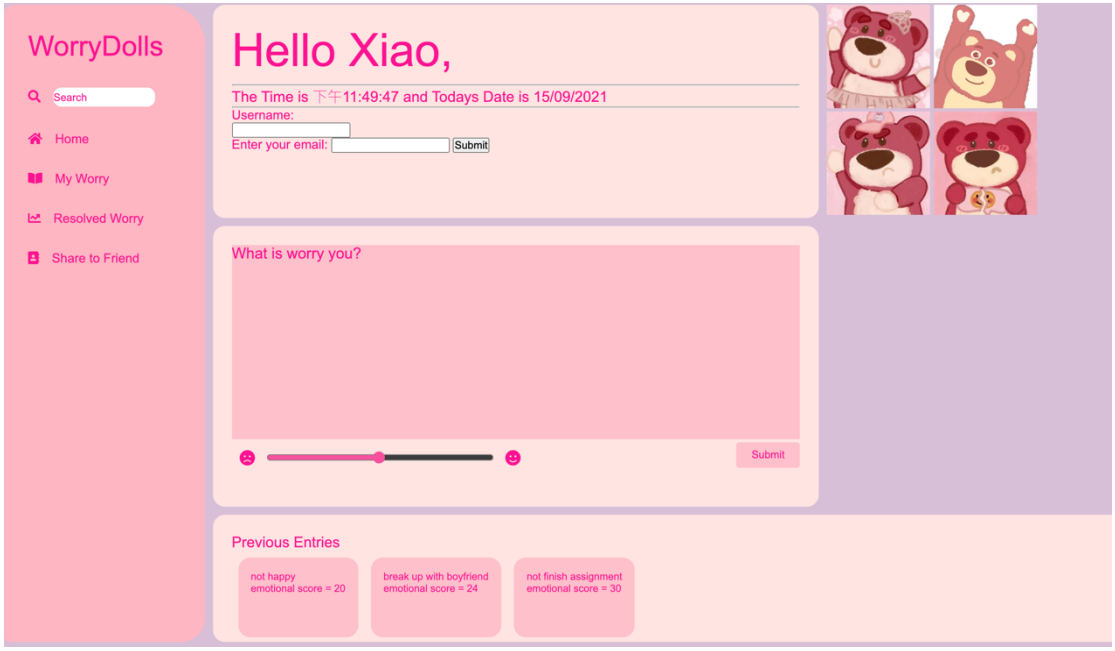
```
<div id="app">
    <!-- Everything on Dashboard -->
    <div class="dash Wrapper">
        <!-- menu  -->
        <div class="menu">
            <!-- menu Title  -->
            <h2 class="menuTitle">WorryDolls</h2>
            <!-- Hamburger menu button with javascript method showMenu() -->
            <a href="javascript:void(0);" v-on:click="showMenu()" class="icon" onclick="myFunction()">
                <i class="fa fa-bars"></i>
            </a>
            <!-- items in menu nav -->
            <div id="menuItems">
                <li><i class="fas fa-search"></i><input class="search" type="text" placeholder="Search"></a></li>
                <li><i class="fas fa-home"></i><a href="index.html">Home</a></li>
                <li><i class="fas fa-book-open"></i><a href="worrys.html">My Worry</a></li>
                <li><i class="fas fa-chart-line"></i><a href="resolved.html">Resolved Worry</a></li>
                <li><i class="fas fa-address-book"></i><a href="share.html">Share to Friend</a></li>
            </div>
        </div>
        <!-- main welcome block -->
        <div class="main block">
            <h1 class="mainTitle">Hello {{ Name }},</h1>
            <hr>
            <h3>The Time is {{displayTime()}} and Todays Date is {{displayDate()}}</h3>
            <hr>
            <form>
                <label for="username">Username:</label><br>
                <input type="text" id="username" name="username"><br>
                <label for="email">Enter your email:</label>
                <input type="email" id="email" name="email">
                <input type="submit" value="Submit">
            </form>
        </div>
```

| User story | Acceptance Criteria |
| --- | --- |
| **As an online trouble recorder, I hope to write down my worry** | • **The user must be able to enter text in the text box.**<br>• **The user must be able to confirm after writing input.** |

| | |
|---|---|
| **As an online trouble recorder, I hope to preserve my emotional state.** | • **Prompt for login or new registration at site launch.**<br>• **Login should take users to their home dashboard.**<br>• **Registration should guide the user through the registration process.**<br>• **The user enters the name, email, and password.**<br>• **Stored credentials.**<br>• **Confirmation email has been sent.**<br>• **Once a new entry is written, users should confirm that they have finished editing it.**<br>• **After editing, the portal site should have online storage and save it to online storage.** |
| **I want to automatically add other information, such as date and time, to my Worry entry** | • **Once the entry is edited, the site must give each entry a time stamp.**<br>• **You should use JS to generate timestamps to get the device time.**<br>• **The site should display this timestamp on all entries.** |
| **I want to remove the worry entry feature.** | • **There is a delete button on each entry.**<br>• **Prompts the user to decide whether to delete.**<br>• **Remove entries from pages and cloud storage.** |
| **I want to be able to search my entries.** | • **The site should have a search function.**<br>• **The search should return specific terms related to the search term.** |
| **I'd like to greet them with my name.** | • **Users must be logged in.**<br>• **The site takes the user name and displays it in HTML format.** |
| **I'd like a menu on the side of the desktop.** | • **Web sites must know screen sizes.**<br>• **Adjust the layout to place the menu on one side of the desktop screen size.** |

**Task3:**
Polishing my proposal See assignment 1 for progress or see.

Project progress:

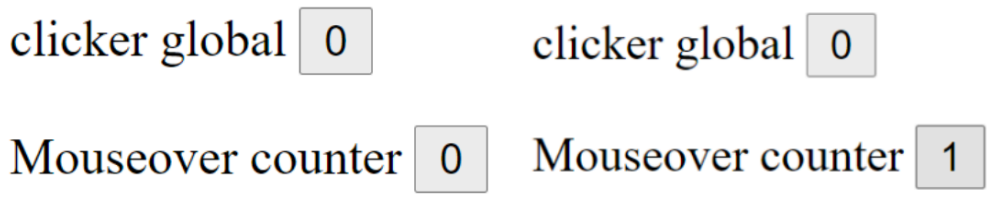



**Week 5:**
Reference:
The goal of this week is to deeply understand and practice the use of components, so that you can more fully understand the functions of Vue as a framework. The key is:
• components are reusable code instances that can be stacked in a tree to effectively decompose the code into manageable parts, so as to avoid repeated editing of a wide range of code when changes are needed.
• components can do many useful things in response, which can make a website more attractive, such as click, mouse movement, key press, etc
• global components are ubiquitous throughout the application. Global registration applies to basic, frequently used components that are needed almost everywhere in the package. The disadvantage is that the component is loaded when the script is loaded, even if it is not required.
• monitors are useful in real-time updates, such as updating search results in real time as users enter.

Task1:

```
//local component that goes to 1 if mouseover and drops back to zero when mous
e moves away
            var mouseCount = {
            template: '<button v-on:mouseover="count++" v-on:mouseout="count--
">{{count}}</button>',
            data () {
                return {
                    count: 0
                }
                }
            }
```

Output:




Task2:

**Runtime + Compiler vs. Runtime-only**

If you need to compile templates on the client (e.g. passing a string to the `template` option, or mounting to an element using its in-DOM HTML as the template), you will need the compiler and thus the full build:

```
// this requires the compiler
new Vue({
  template: '<div>{{ hi }}</div>'
})

// this does not
new Vue({
  render (h) {
    return h('div', this.hi)
  }
})
```

Dev build:

```
git clone https://github.com/vuejs/vue.git node_modules/vue
cd node_modules/vue
npm install
npm run build
```

Task3:

```
<!-- worry block -->
<div class="worry block">
    <!-- worry text input -->
    <textarea v-model="message" class="worryInput" placeholder="What is worry you?"contenteditable="false"></textarea>
    <!-- worry emotional slider -->
    <div class="slidecontainer">
        <i class="fas fa-frown"></i>
        <input type="range" v-model="slider" min="1" max="100" value="50" class="slider" id="myRange">
        <i class="fas fa-smile"></i>
        <button v-on:click="submit">Submit</button>
    </div>
```

Task4:

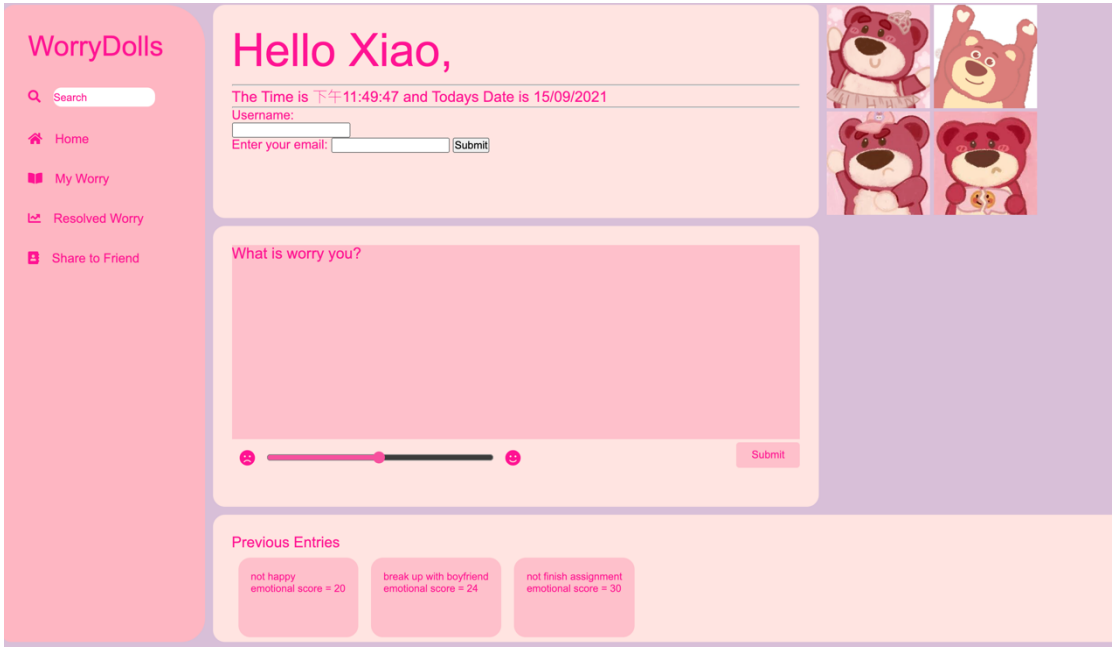
Component Registration

```
<div class="top bar lightpink">
    <button onclick="document.getElementById('id01').style.display='block'" class="log-button log-green log-large">Login</button>
    <form>
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="pwd">Password:</label><br>
        <input type="password" id="pwd" name="pwd">
        <label for="email">Enter your email:</label>
        <input type="email" id="email" name="email">
        <input type="submit" value="Submit">
    </form>
```

Props:

```
<h1 class="mainTitle">Hello {{ Name }},</h1>
<hr>
<h3>The Time is {{displayTime()}} and Todays Date is {{displayDate()}}</h3>
<hr>
```

Project progress:



**Week 6:**

Reference:

In our lecture this week, we first learned what front-end developers are. As a front-end developer, you need to master various skills. Here are some important skills that will help us become front-end developers:

HTML and CSS - the most basic front-end skills.

JavaScript - can add interactivity to our website.

CSS and JS framework - like Vue! Or guide.

Responsive design - you must have a mobile phone first.

Debug - need to be able to find your errors.

Good browser tools - helpful for debugging.

Web Performance - make sure our job is performance.

Command line - useful skills, understanding web servers, etc.

Task1:

```
<form>
    <label for="username">Username:</label><br>
    <input type="text" id="username" name="username"><br>
    <label for="email">Enter your email:</label>
    <input type="email" id="email" name="email">
    <input type="submit" value="Submit">
</form>
```

Task2:

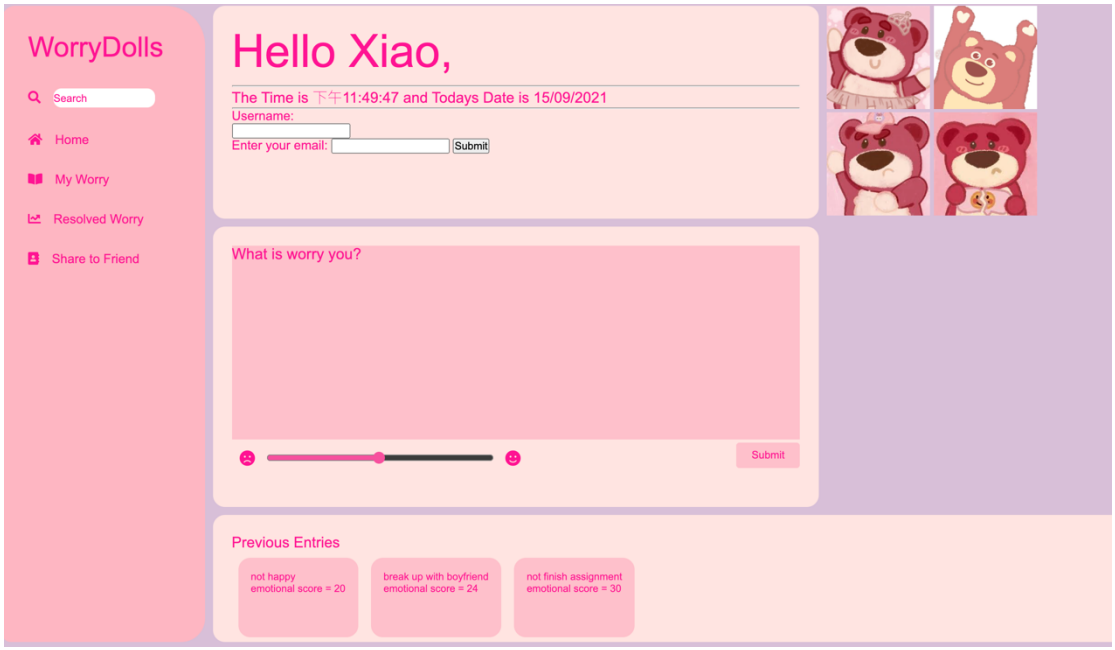
```
<div id="v-model-multiple-checkboxes"
class="demo">
  <input type="checkbox" id="jack"
value="Jack" v-model="checkedNames" />
  <label for="jack">Jack</label>
  <input type="checkbox" id="john"
value="John" v-model="checkedNames" />
  <label for="john">John</label>
  <input type="checkbox" id="mike"
value="Mike" v-model="checkedNames" />
  <label for="mike">Mike</label>
  <br />
  <span>Checked names: {{ checkedNames }}
</span>
```

Task3:

```
<select> ↵
    <option value="Potato option">potato option</option> ↵
    <option value="Pea">Pea</option> ↵
    <option value="Carrot">Carrot</option> ↵
```

Task4:
I do not use use modifiers in my projects.

Project progress:




**Week 7:**

Reference:

This week, we expanded our knowledge of components by introducing some new concepts of using components to provide additional functionality. Slots can contain template code, consisting of HTML or other components. Then we study the difference between dynamic components and asynchronous components. When we switch between tabs, dynamic components will be re rendered, which causes their state to change. If they re render too much, the performance will degrade. However, asynchronous components do not always re render our components, and the results can be cached to improve performance.

Task1:

```
<select> ↵
    <option value="Potato option">potato option</option> ↵
    <option value="Pea">Pea</option> ↵
    <option value="Carrot">Carrot</option> ↵
```

Task2:

```
Vue.component('child',{
  props: {
    text: {
      type: String,
      required: true
    }
  },
  template: `<div>{{ text }}</div>`
});

new Vue({
  el: '#app',
  data() {
    return {
      message: 'hello mr. VUE'
    }
  }
})
```

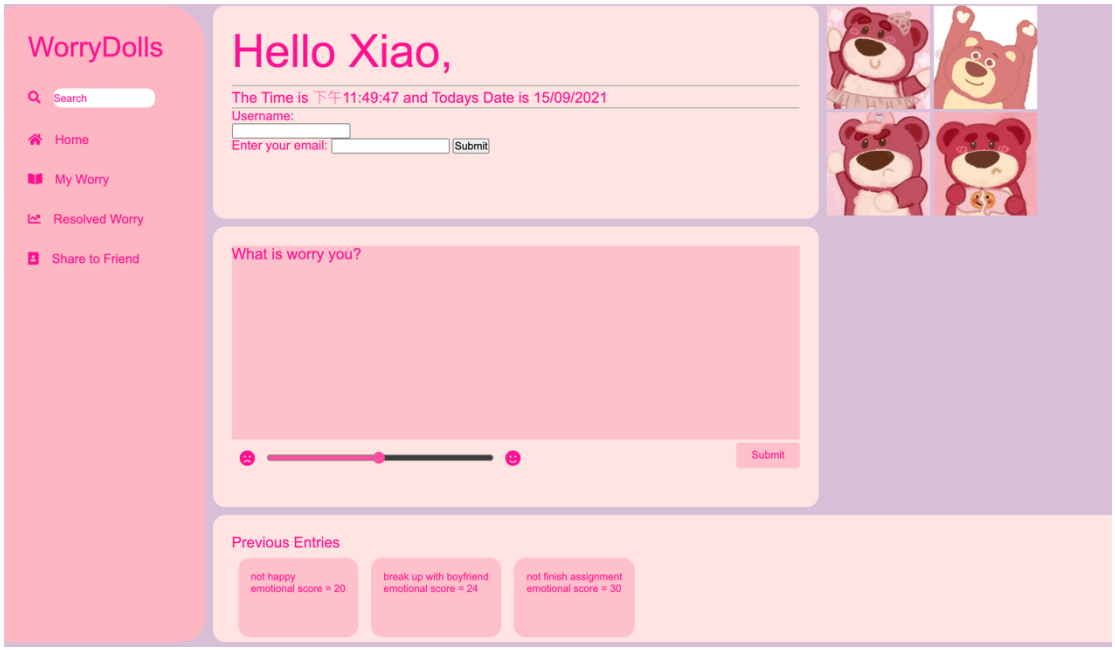
Task3:

```
<form>
    <label for="username">Username:</label><br>
    <input type="text" id="username" name="username"><br>
    <label for="email">Enter your email:</label>
    <input type="email" id="email" name="email">
    <input type="submit" value="Submit">
</form>
```

Task4:

```
<div id="app">
   <section class="hero is-fullheight is-warning is-bold">
     <div class="hero-body">
       <div class="container">
         <p class="title is-1">{{ title }}</p>
         <p class="subtitle is-3">{{ subtitle }}</p>
         <!-- Renders "Hello from the parent!" -->
        <child-component>
          <p slot="bottom">Let's move this into the bottom slot</p>
          <p>Hello from the parent!</p>
          <p slot="top">Let's move this into the top slot</p>
        </child-component>
        <!-- Renders "Hello from the child!" -->
        <child-component></child-component>
       </div>
     </div>
  </section>
</div>

<script type="text/x-template" id="child-component">
  <div>
    <!-- Elements injected with the `slot="top"`
    attribute will end up in here. -->
    <slot name="top">
    </slot>
    <!-- A slot tag without a name is a catch-all,
    it will contain any content that doesn't have
    a `slot=""` attribute. -->
    <slot>
      <p>Hello from the child!</p>
    </slot>
```

Project progress:




GitHub link:
https://github.com/1126xiaofeng/SIT120