Homework 2-1

Part 1:

Codepen's link: https://codepen.io/Wang-Wei-Li/pen/MWRpgZN

Generally, I just move the texts in (<head><style></style></head>) into CSS,

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Shopping List</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
        }

        .container {
            max-width: 600px;
            margin: 50px auto;
            padding: 20px;
            background-color: #fff;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }

        h1 {
            text-align: center;
            margin-bottom: 20px;
        }

        input[type="text"] {
            width: calc(100% - 80px);
            padding: 10px;
            margin-bottom: 10px;
        }
```

```
        button {
            padding: 10px 20px;
            background-color: #007bff;
            color: #fff;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }

        button:hover {
            background-color: #0056b3;
        }

        ul {
            list-style-type: none;
            padding: 0;
        }

        li {
            padding: 10px;
            background-color: #f9f9f9;
            border-bottom: 1px solid #ddd;
            display: flex;
            justify-content: space-between;
        }

        li:last-child {
            border-bottom: none;
        }
    </style>
</head>
```

and the texts in (<body><script></script></body>) into JS.

```
<body>
    <div class="container">
        <h1>Shopping List</h1>
        <input type="text" id="itemInput" placeholder="Add new item">
        <button onclick="addItem()">Add Item</button>
        <ul id="itemList"></ul>
    </div>
    <script>
        function addItem() {
            var input = document.getElementById("itemInput");
            var itemText = input.value.trim();
            if (itemText !== '') {
                var itemList = document.getElementById("itemList");
                var li = document.createElement("li");
                li.textContent = itemText;
                var deleteButton = document.createElement("button");
                deleteButton.textContent = "Delete";
                deleteButton.addEventListener("click", function() {
                    itemList.removeChild(li);
                });
                li.appendChild(deleteButton);
                itemList.appendChild(li);
                input.value = '';
            }
        }
    </script>
</body>
```

Part 2:

Codepen's link: https://codepen.io/Wang-Wei-Li/pen/KKYWKVP

(1) HTML Structure:

The HTML file is simplified to serve primarily as a mounting point ( ) for the React application, emphasizing the use of React to manage the UI.

```
<body>
  <div id="root"></div>
</body>
```

The body of HTML is now rendered through "root" by calling the "ReactShopList" function.

```
render(<ReactShopList />, document.getElementById('root'));
```

(2) State management:

The application's state, including the list of items, is managed using React's useState hook, allowing for more efficient and predictable state updates.

```
const [shops, setshops] = useState([]);
const [newshop, setNewshop] = useState('');
```

(3) JavaScript Functionalities

The "addItem" function in Javascript is divided into several components in React, such as "addshop" and "deleteshop".

```
const addshop = () => {
  const shopText = newshop.trim();
  if (shopText !== '') {
    setshops([...shops, shopText]);
    setNewshop('');
  }
};

const deleteshop = (index) => {
  const updatedshops = [...shops];
  updatedshops.splice(index, 1);
  setshops(updatedshops);
};
```

(4) Return values:

Most texts in <body> of the HTML are now return values of the "ReactShopList" function. I have made some modifications and rearrangements so that they can call the react components (ex: addshop, deleteshop) or just get values from the state constants / array (ex: shop, newshop)

```
  return (
    <div class="container">
      <div id="root">
        <h1>Shopping List (React)</h1>
        <input
          type="text"
          value={newshop}
          onChange={(e) => setNewshop(e.target.value)}
          placeholder="Add new item"
        />
        <button onClick={addshop}>Add Item</button>
        <ul>
          {shops.map((shop, index) => (
            <li key={index}>
              <span>{shop}</span>
              <button onClick={() => deleteshop(index)}>Delete</button>
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
}
```

Part 3:

Codepen's link: https://codepen.io/Wang-Wei-Li/pen/zYXZYKo

(1) HTML structure:

The HTML is simplified to mainly include a mounting point for the Vue application ( ), allowing Vue to handle the application's UI.

```
<body>
  <div id="app"></div>
</body>
```

(2) Vue Application Structure:

The Vue instance is created with a set of options including data, methods, and a template.

data holds the application's state (shops and newshop);

```
data: {
  shops: [],
  newshop: '',
},
```

methods include functions for adding and deleting items from the shopping list.

```
methods: {
  addshop: function() {
    const shopText = this.newshop.trim();
    if (shopText !== '') {
      this.shops.push(shopText);
      this.newshop = '';
    }
  },
  deleteshop: function(index) {
    this.shops.splice(index, 1);
  },
},
```

(3) Template:

Most texts in <body> of the HTML are now put into “template”. There are some noteworthy designs:

a. Vue's v-model directive binds the “newshop” in “data” with the input element.

b. Event handling for adding and deleting items is done using Vue's v-on directive (shorthand @), directly in the template.

c. Vue's v-for directive is used for rendering the list of shopping items, providing an easy way to display an array of items in the template.

```
template: `
<div class="container">
  <div id="app">
    <h1>Shopping List (Vue)</h1>
    <input
      type="text"
      v-model="newshop"
      placeholder="Add new item"
    />
    <button @click="addshop">Add Item</button>
    <ul>
      <li v-for="(shop, index) in shops" :key="index">
        <span>{{ shop }}</span>
        <button @click="deleteshop(index)">Delete</button>
      </li>
    </ul>
  </div>
</div>
`,
```

Homework 2-2

Part 1:

Codepen's link: https://codepen.io/Wang-Wei-Li/pen/rNbyNyM

Generally, I just move the texts in (<head><style></style></head>) into CSS,

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Weather App</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f0f0f0;
        }

        .container {
            text-align: center;
        }

        h1 {
            margin-bottom: 20px;
        }
        input[type="text"] {
            padding: 8px;
            margin-right: 5px;
        }
```

```
        button {
            padding: 8px 20px;
            background-color: #007bff;
            color: white;
            border: none;
            cursor: pointer;
        }

        #weather-info {
            margin-top: 20px;
        }
    </style>
</head>
```

and the texts in (<body><script></script></body>) into JS.

```
<body>
    <div class="container">
        <h1>Weather App</h1>
        <input type="text" id="city-input" placeholder="Enter city name">
        <button id="search-button">Search</button>
        <div id="weather-info"></div>
    </div>

    <script>
        document.getElementById("search-button").addEventListener("click",
        function() {
            var city = document.getElementById("city-input").value;
            if (city.trim() !== "") {
                fetchWeather(city);
            } else {
                alert("Please enter a city name.");
            }
        });

        function fetchWeather(city) {
            var apiUrl = `https://wttr.in/${city}?format=%t+%w+%h`;

            fetch(apiUrl)
                .then(response => {
                    if (!response.ok) {
                        throw new Error("Failed to fetch weather data.");
                    }
                    return response.text();
                })
```

```
        .then(data => {
            displayWeather(data);
        })
        .catch(error => {
            console.error("Error fetching weather data:", error);
            alert("Failed to fetch weather data. Please try again later.");
        });
}

function displayWeather(data) {
    var weatherInfo = document.getElementById("weather-info");
    weatherInfo.innerText = data;
}
    </script>
</body>
```

Part 2:

Codepen's link: https://codepen.io/Wang-Wei-Li/pen/XWQMWOe

(1) HTML Structure:

The HTML file is simplified to serve primarily as a mounting point ( ) for the React application, emphasizing the use of React to manage the UI.

```
<body>
  <div id="root"></div>
</body>
```

The body of HTML is now rendered through "root" by calling the "ReactWeatherApp" function.

```
render(<ReactWetherApp />, document.getElementById('root'));
```

(2) State management:

The application's state is managed using React's useState hook, allowing for more efficient and predictable state updates

```
const [city, setCity] = useState('');
const [weather, setWeather] = useState('');
```

(3) JavaScript Functionalities

Instead of using EventListener, in React we use the onClick event to trigger the weatherInfo constant, which call the fetchWeather function that update the weather's state.

```
<button onClick={weatherInfo}>Search</button>
```

```
const weatherInfo = () => {
  const cityText = city.trim();
  if (cityText !== ''){
    fetchWeather(cityText);
  }
  else {
    alert("Please enter a city name.");
  }
}
```

```
function fetchWeather(city) {
  const apiUrl = `https://wttr.in/${city}?format=%t+%w+%h`;
  fetch(apiUrl)
    .then(response => {
      if (!response.ok) {
        throw new Error("Failed to fetch weather data.");
      }
      return response.text();
    })
    .then(data => {
      setWeather(data);
    })
```

```
<div id="weather-info">{weather}</div>
```

(4) Return values:

Most texts in <body> of the HTML are now return values of the "ReactWetherApp" function. I have made some modifications and rearrangements so that they can call the react components (ex: weatherInfo) or just get values from the state constants (ex: city, weather).

```
  return (
    <div class="container">
      <div id="root">
        <h1>Weather App (React)</h1>
        <input
          type="text"
          value={city}
          onChange={(e) => setCity(e.target.value)}
          placeholder="Enter city name"
        />
        <button onClick={weatherInfo}>Search</button>
        <div id="weather-info">{weather}</div>
      </div>
    </div>
  );
}
```

Part 3:

Codepen's link: https://codepen.io/Wang-Wei-Li/pen/abxWGdV

(1) HTML structure:

The HTML is simplified to mainly include a mounting point for the Vue application ( ), allowing Vue to handle the application's UI.

```
<body>
  <div id="app"></div>
</body>
```

(2) Vue Application Structure:

The Vue instance is created with a set of options including data, methods, and a template.

data holds the application's state (city and weather);

```
data: {
  city: '',
  weather: '',
},
```

methods include functions regarding weather fetching.

```
methods: {
  weatherInfo: function() {
    const cityText = this.city.trim();
    if (cityText !== '') {
      this.fetchWeather(cityText);
    }
    else {
      alert("Please enter a city name.");
    }
  },
  fetchWeather: function(city) {
```

(3) Template:

Most texts in <body> of the HTML are now put into “template”. There are some noteworthy designs:

a. Vue's v-model directive binds the “city” in “data” with the input element.

b. Event handling for calling the “weatherInfo” function is done using Vue's v-on directive (shorthand @), directly in the template.

```
  template: `
  <div class="container">
    <div id="app">
      <h1>Weather App (Vue)</h1>
      <input
        type="text"
        v-model="city"
        placeholder="Enter city name"
      />
      <button @click="weatherInfo">Search</button>
      <div id="weather-info">{{weather}}</div>
    </div>
  </div>
  `,
});
```