CW1 solutions (and CW2 hints)

Solutions to some of the more difficult points in CW1:

- Based on the marking and feedback of CW1.
- Not enough time to cover everything.

Most of these are covered in the lectures already.

- For all the coursework, the lecture slides and the reading linked at the end should be all you need.

CW2 will build on CW1, so most of the features in CW1 will be needed in CW2:

- Without these, you will not be able to get mark in CW2;

Will also discuss points related to CW2.

CW1 requirements recap

Users (9%):

- The app must have two types of users (3%):
    1. The users that look for after-school activities, such as students or their parents,
    2. The users that provide the after-school activities (service providers);
- Both types of users must be able to
    1. register (3%),
        - with at least email address and password;
        - with html email validation;
        - check if the email address is already registered.
    2. login/logout (3%),
        - Display user information on the page, such as their email address, when login successfully;
        - Provide error information whether the password is wrong or the email is not registered;
        - Remove all the user information from the page once log out.

For student and parents, they should be able to (12%):

- Search for classes and activities by keyword (3%);
    - The search will be on activity topic (such as 'math' or 'sport club') only, and not on other fields such as price or review;
    - The search must return all the activities whose topic contains the search term. For example, both 'math' and 'sports club' should be returned if the search term is a single letter 't';
    - Search must also work on the result of filtering.
- Filter results by topic, price, or review (3%);
    - Filter can be applied more than once. For example, first by topic and then by price;
    - Filter must work on the result of search, i.e., can filter search results.
    - Only available options are shown in the filter menu: for example, if there is no 'math' class, 'math' should not be shown in the 'topic' filter.

- Sort the results by topic (alphabetically), price range, or review (3%);
    - Sorting by one attribute (such as 'price') only; selecting a new sorting criteria (such as 'topic') will replace the previous sorting criteria;
    - There must be options to sort in ascending or descending order;
    - Sorting must work on the result of search or filtering.
- Leave reviews for class or activity (3%).
    - The only review needed is a ranking between 1 and 5 stars;
    - An user can only leave review once for a class or activity and cannot change afterwards;
    - The display of each class or activity should include the average rating and number of reviews.

In addition, for service provider, they should be able to add/update/remove class and activities (4%).

- A service provider should see a list of all his/her classes and activities once logged in;
- Each class or activity listing should have at least the following information:
    - Topic, such as 'math' or 'sports club';
    - Price, i.e., total cost;
    - Location, where the class/activity will happen;
    - Time and length, i.e., when is the class/activity (such as 4pm every weekdays) and how long it is (such as 1 hour).
- A service provider should be able to add new class/activity or edit the information of existing classes/activities.

Solutions (one of the many possible ways)

Step 1: design you app

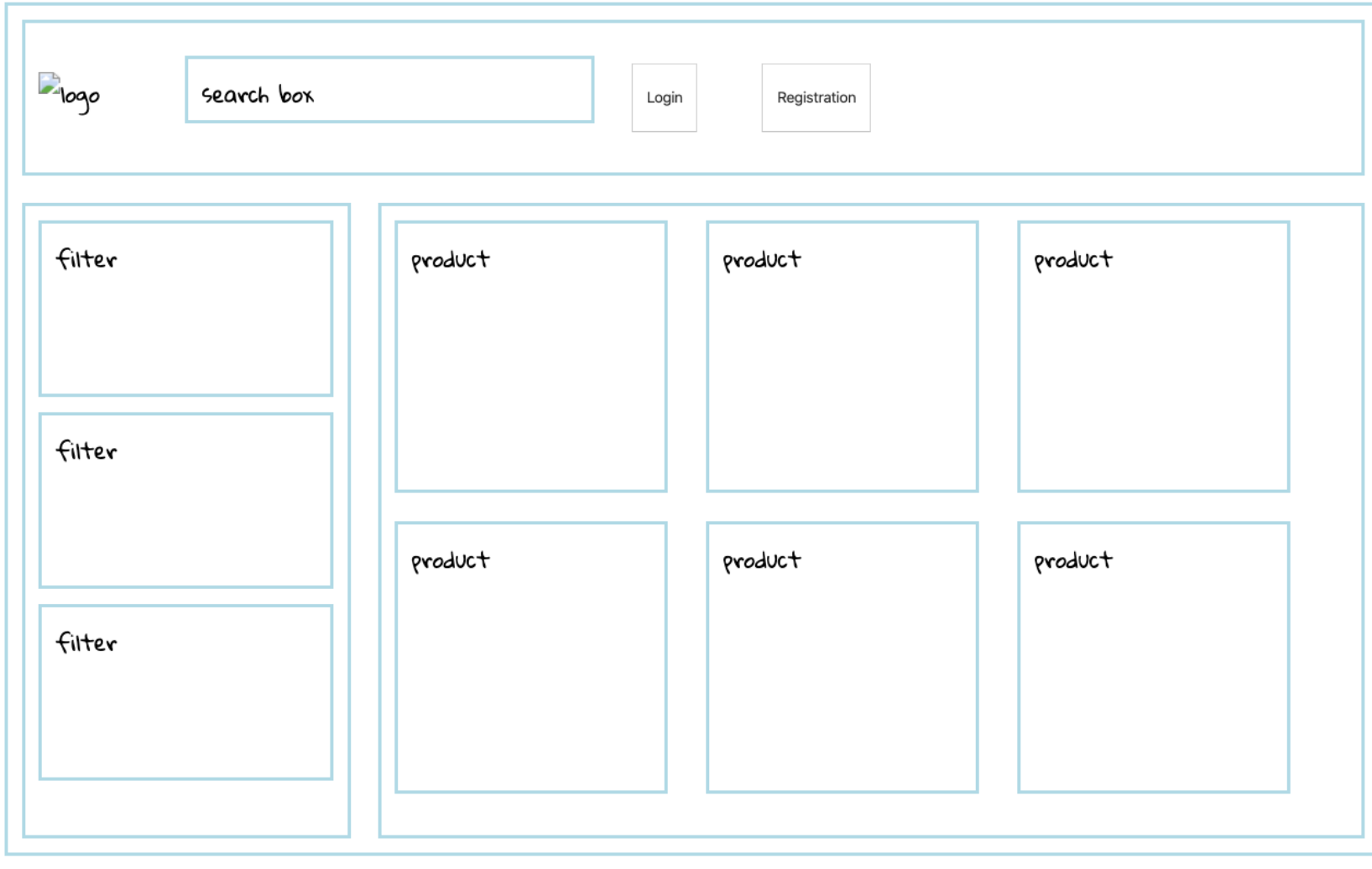
How many pages

two pages

- one page for search/filter and user registration/login;
- one page for service provider creating and editing lessons.

What each page looks like

Use figma, justinmind, or any other wireframe tool (week 4 lecture):


logo
search box
Login
Registration
filter
filter
filter
product
product
product
product
product
product

Step 2: create the HTML and CSS

This was covered in the 2nd year and Week 4 lecture (CSS grid layout).

```html
<!DOCTYPE html>
<html lang="en">
<head>
    ...
    <link rel="stylesheet" href="style2.css">
</head>

<body>
    <div class="container">
        <header>
            <img id='logo' src="" alt="logo">
            <div id='searchbox'>search box</div>
            <button>Login</button>
            <button>Registration</button>
        </header>
        <aside>
            <div class="filter">filter</div>
            <div class="filter">filter</div>
            <div class="filter">filter</div>
        </aside>
        <main>
            <div class="product">product</div>
            <div class="product">product</div>
            <div class="product">product</div>
            <div class="product">product</div>
            <div class="product">product</div>
            <div class="product">product</div>
        </main>
    </div>
</body>

</html>
```

The CSS

```css
@import url('https://fonts.googleapis.com/css?family=Gloria+Hallelujah&display=swap');

.container {
    display: grid;
    grid-template-columns: 1fr 3fr;
    gap: 20px;
    font-family: 'Gloria Hallelujah', cursive;
}

div,header,aside,main {
    border: solid lightblue;
    box-sizing: border-box;
    padding: 10px;}

header {
    grid-column: 1/3;
    grid-row:1;
}

#logo {width: 100px;}

#searchbox {width: 300px;}

#logo, #searchbox {
    height: 50px;
    display: inline-block;}

button {
    height: 50px;
    margin: 20px;
    padding: 10px;}

aside {
    grid-column: 1;
    grid-row:2;
}
```

Storing data

All data that need to be kept after page refresh need to store in local storage:

- user information: email, password
- lessons: topic, price, location, time

Probably as two arrays of json objects.

```
users: [
    {"email": "e1@address.com", "password": "1234"},
    {"email": "e2@address.com", "password": "1234"},
    ...
]
```

```
lessons: [
    { 'topic': 'math', 'location': 'hendon', 'price': 100 },
    { 'topic': 'math', 'location': 'colindale', 'price': 80 },
    ...
]
```

We will add more later when needed.

Users (9%):

- The app must have two types of users (3%):
    1. The users that look for after-school activities, such as students or their parents,
    2. The users that provide the after-school activities (service providers);
- Both types of users must be able to
    1. register (3%),
        - with at least email address and password;
        - with html email validation;
        - check if the email address is already registered.
    2. login/logout (3%),
        - Display user information on the page, such as their email address, when login successfully;
        - Provide error information whether the password is wrong or the email is not registered;
        - Remove all the user information from the page once log out.

Sign in and sign up

Different buttons and fields need to be shown/hidden depends on the app status:

- When the app is first open, there should be option to either `sign in` or `sign up`:
    - `email` and `password` field
    - `sign in` and `sign up` button
- Once a user signed in,
    - both `sign in` and `sign up` button should disappear;
    - `sign out` button should appear
    - user email should be displayed on the page.

These can be achieved with `v-if` :

- add all the html elements;
- use `v-if` to selectively hide some of them.

HTML: add all the element

```
<div id="userform">
    <span v-if='currentUser'>
        {{currentUser}}
        <button @click='signout'>Sign out</button>
    </span>
    <span v-else>
        email <input v-model='email'>
        password <input type='password' v-model='password'>
        <button v-on:click='signin'>Sign in</button>
        <button v-on:click='signup'>Sign up</button>
        {{userMessage}}
    </span>
</div>
```

Vue instance

```
var userform = new Vue({
    el: "#userform",
    data: {
        currentUser: '',
        email: '',
        password: '',
        userMessage: ''
    },
    methods: {
        signup: function() {...}
        signin: function() {...}
        signout: function() {...}
    }
})
```

Sign up function

```
signup: function () {
    this.userMessage = '';

    // more validation should be included
    if (!this.email) {
        this.userMessage = 'error: empty email';
        return
    }
    if (!this.password) {
        this.userMessage = 'error: empty password';
        return;
    }
    var newUser = {
        'email': this.email,
        'password': this.password
    }

    // a user array should be used
    localStorage.setItem('users', JSON.stringify(newUser))
},
```

Common error: account creates even when the email validation fails

- html5 email validation does not stop the `signup` function
- such email validation needs to be implemented in the `signup` function

Check duplicate email (week 7 lecture)

```
var users = '';
var newEmail = this.email;
if (localStorage.getItem('users')) { // 'users' is an array of objects
    users = JSON.parse(localStorage.getItem('users'));
};
if (users) {
    if (users.some(function (user) { return user.email === newEmail })) {
        alert('Email already exists!');
        return;
    }
    users.push({ 'email': newEmail, 'password': this.password });
    localStorage.setItem('users', JSON.stringify(users));
}
else {
    users = [{ 'email': newEmail, 'password': this.password }];
    localStorage.setItem('users', JSON.stringify(users));
}
```

Two types of users

Use a radio button to select user type when sign up

```
...
<input type="radio" id="option1" value="student" v-model="userType">
<label for="option1">Provider</label>
<input type="radio" id="option2" value="provider" v-model="userType">
<label for="option2">Provider</label>
...
```

The user type is then saved in the user object:

```
users: [
    {"email": "e1@address.com", "password": "1234", "type":"student"},
    {"email": "e2@address.com", "password": "1234", "type":"provider"},
    ...
]
```

Sign in and sign out

```
signin: function () {
    var savedUser = JSON.parse(localStorage.getItem('users'));
    if (savedUser.email == this.email && savedUser.password == this.password) {
        this.currentUser = this.email;
    }
    else {
        this.userMessage = ' error: username or password is not correct.';
        return;
    }
},

signout: function () {
    this.currentUser = '';
    this.email = '';
    this.password = '';
}
```

Search for classes and activities by keyword (3%);

- The search will be on activity topic (such as 'math' or 'sport club') only, and not on other fields such as price or review;
- The search must return all the activities whose topic contains the search term. For example, both 'math' and 'sports club' should be returned if the search term is a single letter 't';
- Search must also work on the result of filtering.

Course list (`courses.js`)

```
var courses = [
    { 'topic': 'math', 'location': 'hendon', 'price': 100 },
    { 'topic': 'math', 'location': 'colindale', 'price': 80 },
    { 'topic': 'math', 'location': 'brent cross', 'price': 90 },
    { 'topic': 'math', 'location': 'golders green', 'price': 120 },
    { 'topic': 'english', 'location': 'hendon', 'price': 110 },
    { 'topic': 'english', 'location': 'colindale', 'price': 90 },
    { 'topic': 'english', 'location': 'brent cross', 'price': 90 },
    { 'topic': 'english', 'location': 'golders green', 'price': 130 },
    { 'topic': 'piano', 'location': 'hendon', 'price': 120 },
    { 'topic': 'piano', 'location': 'golders green', 'price': 140 }
]
```

HTML

```
<div id="search">
    Search <input v-model='term'><br>
    <div v-for='course in results' class='course'>
        <ul>
            <li v-for='(value, key) in course'>
                {{key}}: {{value}}
            </li>
        </ul>
    </div>
</div>
```

Vue

```
var searchApp = new Vue({
    el: '#search',
    data: {
        'term': ''
    },
    computed: {
        results: function () {
            return courses.filter(course =>
                course.topic.includes(term)
            )
        }

    }
});
```

Search a

- topic: math
- location: hendon
- price: 100

- topic: math
- location: colindale
- price: 80

- topic: math
- location: brent cross
- price: 90

- topic: math
- location: golders green
- price: 120

- topic: piano
- location: hendon
- price: 120

- topic: piano
- location: golders green
- price: 140

Filter results by topic, price, or review (3%)

- Filter can be applied more than once. For example, first by topic and then by price;
- Filter must work on the result of search, i.e., can filter search results.
- Only available options are shown in the filter menu: for example, if there is no 'math' class, 'math' should not be shown in the 'topic' filter.

Example from week 7 lecture

```
<div id="filter">
    <h1>Filters</h1>
    <select multiple v-model='userLocations'>
        <option v-for='topic in locations'>
            {{topic}}
        </option>
    </select>

    Selected locations: {{userLocations}}

    <button v-on:click='reset'>reset</button>

    <h1>Course list</h1>
    <div v-for="course in results" class="course">
        <ul>
            <li v-for='(value, name) in course'>
                {{name}}: {{value}}
            </li>
        </ul>
    </div>
</div>
```

Vue

```
var filterApp = new Vue({
    el: '#filter',
    data: {
        userLocations: []
    },
    methods: {
        reset: function () {
            this.userLocations = [];
        }
    },
    computed: {
        locations: function () {
            // return an array of all the locations
            return [...new Set(courses.map(x => x.location))]
        },
        results: function () {
            return courses.filter(course =>
                this.userLocations.length == 0 ||
                this.userLocations.includes(course.location)
            )
        }
    }
})
```

Filters

hendon
colindale
brent cross
golders green

Selected locations: ["colindale"] reset

Course list

- topic: math
- location: colindale
- price: 80

- topic: english
- location: colindale
- price: 90

Combining search and filter

Make search and filter condition separate boolean values.

Combine them when filtering the course list.

This way more filters can be easily added using the same pattern.

HTML

```
<div id="searchFilter">
    <!-- search -->
    Search <input v-model='term'><br><br>

    <!-- filter -->
    <select multiple v-model='userLocations'>
        <option v-for='location in locations'>
            {{location}}
        </option>
    </select>
    <button v-on:click='reset'>reset</button><br>

    <!-- results -->
    <div v-for='course in results' class='course'>
        <ul>
            <li v-for='(value, key) in course'>
                {{key}}: {{value}}
            </li>
        </ul>
    </div>
</div>
```

Vue

```
var searchApp = new Vue({
    el: '#searchFilter',
    data: {
        term: '',
        userLocations: []
    },
    methods: {
        reset: function () {
            this.userLocations = [];
        }
    },
    computed: {
        locations: function () {
            // update option list
            return [...new Set(this.results.map(x => x.location))]
        },
        results: function () {
            return courses.filter(course => {
                // search condition
                var searchCourse = course.topic.includes(this.term);

                // filter condition
                var filterCourse = this.userLocations.length == 0 ||
                this.userLocations.includes(course.location);

                // combine the result
                return searchCourse && filterCourse;
            })
        },
    },
});
```

Search p

hendon
golders green

reset

- topic: piano
- location: hendon
- price: 120

- topic: piano
- location: golders green
- price: 140

Sorting

Use the `sort()` function of the JavaScript array.

Sort the `results` array, so it works on the results of search and/or filtering.

Rating

Add a ratio button with 1-5 rating to each course.

After submission, the rating needs to be saved in local storage, either with `users` or `lessons` array.

The information is then used to

- display total review number,
- calculate average rating, and
- stop same user from rating again.

```
lessons: [
    { 'topic': 'math', 'location': 'hendon', 'price': 100, 'ranking': [
        {'user': 'email1', 'ranking': 5},
        {'user': 'email2', 'ranking': 3},
        ...
    ]},
    ...
]
```

Service provider

Each lesson will need a `provide` property, so each provider only sees his or her courses:

```
lessons: [
    { 'topic': 'math', 'location': 'hendon', 'price': 100, 'ranking': [...], 'pro
vider': 'email'},
    ...
]
```

Adding a new course or editing an existing course can be achieved with forms:

- each property, such as 'topic' or 'location', is a text field,
- when creating a course, each field starts empty,
- when editing a course, each field starts with current value,
- the form data binding and the editing/submission handler can be done with `v-model` and `v-on` respectively.