Programming for the Web (CS-544-01)
Homework – 4

Name – Chaitanya Kulkarni **B-Number: B00814455**

Ans 1 –

One of the possible reasons is that, since the problem is of increase in size of memory footprint of express.js server, we can assume that the express.js server must be using the in-memory sessions. i.e. servers own RAM or local memory. And when this memory is completely filled, it starts memory leakage and causes the server to crash. Hence, the company needs to restart the server. In process of restarting, the memory is cleared out and erased and due to which the contents of shopping carts are lost after the restart, but the application is now up and running.

Ans 2 –

The data model can be prepared as below -

```
const datamodel = {
  studentName: data.students.b123456.name,
  studentID: Object.keys(data.students)[0],
  grades:{
    rubricCategories: {
      hw: {
        id: data.rubrics.hw4.category,
        grade: data.grades.hw4.b123456.points,
        average: 'Calculate the average by iterating through points of that rubric categor
y',
      },
      prj: {
        id: data.rubrics.prj4.category,
        grade: data.grades.prj4.b123456.points,
        average: 'Calculate the average by iterating through points of that rubric categor
y',
      },
    },
  },
  average: 'Calculated by iterating through all the points in all the rubric category',
  rank: 'Determined using the overall grade assigned to that student',
};
```

As per the requirements, the model consists of studentName, studentID, grades - organized by rubric category and by rubric id within each rubric. Also having the average of every rubric category. The model also consists of Total average of all the rubrics and the rank of that student. We can pass this model to the mustache template which will show the required data of that student on HTML.

Ans 3 –

```
<head>
   <script src="https://code.jquery.com/jquery-3.5.0.js"></script>
</head>
<body>
<h1>Grades for a Student</h1>
<h2 id="details" style="color:red;"></h2>

<dl>
   <dd class="students">
```

```
      <dl class="grades">
         <dt>Student Name - </dt>
         <dd id="name"><strong>Harry Potter</strong></dd><br>
         <dt>Homework Grades</dt>
         <dd id="homework">
            <ul>
               <li id="homework1"><label>Homework 1 :<label> <p>94</p></li>
               <li id="homework2"><label>Homework 2 :<label> <p>95</p></li>
               <li id="homework3"><label>Homework 3 :<label> <p>99</p></li>
               <li id="homework4"><label>Homework 4 :<label> <p>100</p></li>
            </ul>
         </dd>
       <dt>Project Grades</dt>
        <dd id="project">
            <ul>
               <li id="project1"><label>Project 1 :<label> <p>100<p></li>
               <li id="project2"><label>Project 2 :<label> <p>90<p></li>
               <li id="project3"><label>Project 3 :<label> <p>95<p></li>
               <li id="project4"><label>Project 4 :<label> <p>100<p></li>
            </ul>
         </dd>
      </dl>
   </dd>
</dl>

<script type="text/javascript">
const fns = (function() {
const fns = {};
const select = fns.select = function ($elements) {
   //Just making other elements back to normal white background
   //for proper understanding
   makeElementsNormal();
   //Highlight the selected element
   $elements.css('backgroundColor', 'yellow');
}
function makeElementsNormal(){
   $('dd #name').css('backgroundColor', 'white');
   $('#homework1').css('backgroundColor', 'white');
   $('#project4').css('backgroundColor', 'white');
   $('#details').html(``);
}
function getScore($element, isAvg){
   makeElementsNormal();
   let gradesArr = [];
   let sum = 0;
   for ( let i = 0; i < $element.length; i++ ) {
      gradesArr.push(parseInt($element[i].innerHTML));
   }
   if(isAvg){
      const arrSum = gradesArr => gradesArr.reduce((a,b) => a + b)
      const arrAvg = gradesArr => gradesArr.reduce((a,b) => a + b) / gradesArr.length;
      $("#details").html(`Average of all Homeworks = ${arrSum(gradesArr)}/${gradesArr.leng
th} = ${arrAvg(gradesArr)}`);
   }
   else{
      $("#details").html(`Homework Grades are : [${gradesArr}]`);
   }
}

fns.f1 = () => select($('dd #name'));
fns.f2 = () => getScore($('#homework ul li p').toArray());
fns.f3 = () => getScore($('#homework ul li p').toArray(), 1);
fns.f4 = () => select($('#homework1'));
fns.f5 = () => select($('#project4'));
return fns;
})();
</script>

<style rel="stylesheet" type="text/css">
li p {
    display: inline;
}
</style>
```

a. **Adding a class hightlight to the element on the page which contains the student name.**
   `$('dd #name')` will select the element <dd> having id = name for highlighting
b. **Returning a jquery list containing all the elements on the page which contain homework grades.**
   `$('#homework ul li p').toArray()` selects the <p> elements inside the unordered list and gets the grades in array.
c. **Calculating the average of all the homework grades contained on the page.**
   `$('#homework ul li p').toArray()` similar to above, this will select the <p> tag from where we can fetch the list to calculate sum and average.
d. **Adding a hightlight class to the element on the page which contains the grade for the first homework.**
   `$('#homework1')` selects the element having id=homework1 for highlighting the same.
e. **Identifying the element on the page which contains the grade for prj4.**
   To identify the element that contains grades for project 4, we can use the selector that has id=project4 like - `$('#project4)`. Here I have identified the same and have highlighted it to recognize clearly.

Ans 4 –
a.)
Advantages of using this approach -
- It allows Virtual DOM, i.e. the virtual DOM is the core reason why React enables the creation of fast, scalable web apps. Through React's memory reconciliation algorithm, the library constructs a representation of the page in a virtual memory, where it performs the necessary updates before rendering the final webpage into the browser
- Server-side rendering in react is very fast and efficient
- Makes the use of JSX - ReactJs introduces special JavaScript code, pre-processor step adding XML-like syntax into JavaScript. It helps to conduct building readable code, to save it in one verified file.

Disadvantage/ Major security problem -
- Using the web service approach makes any application vulnerable to any hacker by using server-side rendering attacker. It results into XSS vulnerability. Cross site scripting (XSS) is a potentially serious client-side vulnerability. It happens whenever an attacker can trick a website into running JavaScript in their target's browser.

- That is any end user may manipulate the data at their end and also there is possibility that the attacker might fetch the potentially secured data at client side, resulting into unwanted hacking of user information.
- To overcome this problem, all user input should have HTML entities escaped. When you are using React, it does prevent most XSS vulnerabilities, due to how it creates DOM Nodes and textual content.
- `$ npm install --save serialize-javascript`
- When serializing state on the server to be sent to the client, you need to serialize in a way that escapes HTML entities. This is because you are often no longer using React to create this string, hence not having the string automatically escaped. Hence to use this, we can install npm module serialize-JavaScript to prevent the XSS vulnerability.

b.)

The react component I have used here is named as StudentGrades. Which consist of a form that takes the studentID as user input and upon submit, it displays the respective student data on that page. The details of the design are as follows -

- Properties of the components:

- Props of the component will consist of the this.state object which will store the the value of the studentID field entered by the user and the this.state.studentInfoArray which is an array of object consisting information about grades of particular student.
- code segment:

```
constructor(props) {
          super(props);
          this.state = {value: ''};
          this.handleChange = this.handleChange.bind(this);
          this.handleSubmit = this.handleSubmit.bind(this);
          this.state.studentInfoArray = [];
        }
```

- How will the component access the web service data:

- Once the submit button is clicked, it will call the respective handleSubmit method and the data is fetched by web-service, we store that data in the studentInfoArray mentioned above using this.setState() method of react.
- Code segment:

```
this.setState(prevState => ({
    studentInfoArray: [{
        studentName: data.students[this.state.value].name,
        studentID: this.state.value,
        grades:{
            rubricCategories:{
            hw: [{
                id: data.rubrics.hw4.category,
                grade: data.grades.hw4[this.state.value].points,
                average: 'Calculate the average by iterating through points of that rubric
 category',
            }],
            prj: [{
                id: data.rubrics.prj4.category,
                grade: data.grades.prj4[this.state.value].points,
                average: 'Calculate the average by iterating through points of that rubric
 category',
            }],
            },
        },
        average: 'Calculated by iterating through all the points in all the rubric categor
y',
        rank: 'Determined using the overall grade assigned to that student',
    }]
}));
```

- A description of the massaging which should be done on the data in order to have the render() function contain simple jsx.

- In order to render the data on the html, we use the JSX script. This is returned in the render() function. This consists of the html code that is to be rendered. Since we store the data in the this.state property of component, we can fetch the data from the same.
- We need to write a proper HTML script in order to render the data on web page. We manipulate HTML using the data structure obtained from the web-service.
- We use map() function to iterate over the array studentInfoArray to display the data of that student. Hence, Massaging of the data can be done as below:

```
<div>
    {this.state.studentInfoArray.map(student => (
    <div className="student" key={student.studentID}><br />
        Student ID: {student.studentID}<br />
        Name: {student.studentName}<br />
        <div>
            &emsp;Grades<br />
            {student.grades.rubricCategories.hw.map(rubric=>(
                <div key={rubric.id}>
                    &emsp;&emsp;Category: {rubric.id}<br />
                    &emsp;&emsp;grade: {rubric.grade}<br />
                    &emsp;&emsp;average: {rubric.average}<br />
                </div>
            ))}
        </div>
        <br />
        <div>
            {student.grades.rubricCategories.prj.map(rubric=>(
                <div key={rubric.id}>
                    &emsp;&emsp;Category: {rubric.id}<br />
                    &emsp;&emsp;grade: {rubric.grade}<br />
                    &emsp;&emsp;average: {rubric.average}<br />
                </div>
            ))}
        </div>
        Average: {student.average}<br />
        Rank: {student.rank}<br />
    </div>
    ))}
</div>
```

Ans 5 –

a. **A control which allows the user to provide the percentage of components which are defective.**
We can use input tag type range (slider) and a simple text box (user can enter number) to get the percentage between 0 to 100.
```
<input type="range" id="percentage" name="percentage" min="0"
max="100"> or

<input name="percentage" type="number" min="0" max="100">
```

b. **A control which allows the user to select one-or-more of the courses which are currently being offered by the CS department.**
We can user Multi-select attribute to select one-or-more courses using <select> tag.
```
<select name="courses" multiple>
<option value="500">CS 500</option>
<option value="545">CS 545</option>
<option value="570">CS 570</option>
<option value="580">CS 580</option>
</select>
```

c. **A control which allows the user to enter a customer-service complaint.**
Service complaint can be a large or small. Hence, we can use text area to put complaints.
```
<textarea name="complaint" rows="10"></textarea>
```

d. **A control which allows the user to enter in a BU B-Number.**
BU B-number consists of a string starting with B and followed by 8 characters. We can check this using regex in input tag. We can show sample example using a placeholder
```
<input name="bNumber" pattern="B\d{8}" placeholder="B00814455">
```

e. **A control which allows the user to select their favorite dessert from a predefined list of desserts.**
We can either use select box that select only one choice or radio-button that allows to select only 1 option from multiple entries.
```
<select name="favoriteDessert">
<option value="cake">Cake</option>
<option value="icecream">Ice-cream</option>
...
</select>
```

Or Radiobutton as –
```
<input type=”radio” id="cake" value=”cake”> - <label>Cake</label>
<input type=”radio” id="" value=”pie”> - <label>Pie</label>
```

f. **A control which allows a US user to enter in a telephone number (which may be an international phone number).**
To enter phone number, we can use textbox input with regex constraints for 10 digits with separating hyphen (-) for area codes. We can place a placeholder for proper understanding. Since, US user is trying to enter the number, he may not know the international area code for specific country. Hence, we can leave it upto them to enter it.
```
<input   name="phNumber"   type="tel"   pattern="^([0-9]+?(-?\d{3})-
?)?(\d{3})(-?\d{4})$" placeholder="1-223-364-7684">
```

Ans 6 –

A. **FALSE –** props of Reactjs component cannot be mutated to send information to its containing component. Properties are immutable during the lifetime of a component.

B. **FALSE –** In HTML, there are some tags like <input>, etc that can be open but not always closed. These are unbalanced tags. Therefor the nested syntax can break because of such tags. Hence false.

C. **TRUE –** XML markup language must be properly nested. An XML document is called well-formed if it satisfies certain rules, specified by the W3C - Nesting of elements within each other in an XML document must be proper.

D. **TRUE –** X in JSX stands for XML; Hence it should always follow the same rule specified by w3c in order to be well-formed. So, if JSX is properly nested, the JSX processor can identify the end of the JSX expression.

E. **TRUE –** To never see the click event, we can use JS's inbuilt method called Event.preventDefault(); The Event interface's preventDefault() method tells the user agent that if the event does not get explicitly handled, its default action should not be taken as it normally would be. i.e. the preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.