CIS 3308 Client Side Challenge

Functional Requirements

Write an HTML page that:

- Allows the user to make an AJAX call that returns all the (JSON-ized) data, from either your user table or your other table (from your database). At least one of the fields from your table must be non-character and null-able.
- Builds a user interface to show the data list (HTML table, or divs, etc).
- Allows the user to sort the data list by the various fields (by age, by name, etc).
- Allows the user to add a new record to the data list (client side only, no AJAX call), using client side validation (such that these records will be acceptable to insert/update into the database).
- Allows the user to delete any record from the data list by clicking on a delete icon that is associated with that record (client side only, no AJAX call).
- Allows the user to store the data list to a cookie.
- Allows the user to read the data list from a cookie.
- Allows the user to make an AJAX call that sends all the client side data for the purpose of updating your database.

Here is an example User Interface:

In the example above, Age is an optional numeric field (please keep in mind that storing age to a database is not a good idea, birth day or year would be better). Also, you see one record has an id and another does not. Newly created records on the client side would not have IDs and these would be the records to be inserted when the user dumps the data to the database.
NOTE: you can skip one of these requirements and still get full credit:

- database write,
- client side sorting,
- client side deleting.

Design Specifications

- **AJAX calls:** put all your AJAX code in a separate JavaScript file and reference that using a `<script>` tag from your HTML page. You'll need an ajax function where you pass it a URL and the name of your call back function. When your ajax function invokes your call back function, it should pass back an object (that was eval-ed from the http response) and a status/error message. The ajax call (from your HTML page, to your ajax function "Ajax" in your JavaScript page) might look like this:

  ```javascript
  function ajaxCall() {
    Ajax("personList.json", personSuccess, personError);
  }

  function personSuccess(responseObj) {
    console.log("ajax get person list success: ");
    console.log(responseObj);
    personList = responseObj.personList;
    displayPersons();
  }

  function personError(status) {
    my$("status").innerHTML = status;
  }

  } // ajaxCall
  ```

- **Sort.** Don't "reinvent the wheel" here. Google "JavaScript sort" and find an example that uses the built-in JavaScript method: `array.sort(arrayToBeSorted, comparatorFunction)`. The comparator function can be defined in-line (inside the call to array.sort) – it takes two array elements as input and returns a 1, 0, or -1 to indicate the sort order between the two elements.

- **Adding a record** (to the client side data list). Google something like "JavaScript add array element". There is a JavaScript built-in method: `array.push(elementToBeAdded)`. 

- **Validation.** Take advantage of HTML5 validation by surrounding all the user inputs by a form tag with an id attribute (but no other attributes). Inside the form, put a submit button that has an id but is invisible (style="display:none"). When the user clicks the "Add" button (input type="button", not "submit"), a function like this can be called:

  ```javascript
  function addContact() {
    if (!document.getElementById('contactForm').checkValidity()) {
      document.getElementById('contactSubmit').click();
      console.log("not valid");
    } else {
      // user input passed HTML5 validation, proceed with adding an object to your data list array ...
    } // addContact
  ```
• **Cookies.** Put all your cookie code in a separate JavaScript file and reference that using a `<script>` tag from your HTML page. Google "JavaScript read from cookie, write to cookie". There is some sample code from w3Schools, borrow that. This code will read a write a string to a cookie. Enhance that JavaScript Cookie code to provide two new functions like writeCookieObj and readCookieObj – you can call the methods (that you lifted from the internet) but add your own JSON to object conversion using
  o `JSON.stringify(object)` and
  o `eval('(' + JSON_string + ')')`

• **Delete.** When building the UI, add an onclick function to a delete icon that is inside of the visual representation of an element of your data list. Google "JavaScript delete from array". Use `array.splice(startingIndexForDelete, howManyItemsToDelete)`.

  ```javascript
  var deleteIcon = document.createElement("img");
  deleteIcon.src = "delete.png";
  deleteIcon.index = i; // you can add a custom attribute to a DOM element, e.g. array index
  contactHeader.appendChild(deleteIcon);

  deleteIcon.onclick = function () {
    console.log("to delete person " + i);

    // "this" means what is being clicked...
    personList.splice(this.index, 1); // delete 1 element starting at position "this.index"
    this.parentNode.parentNode.style.display = "none";
    displayPersons(); // rebuild UI from current person list, associate new person index to each delete icon.
  }
  ```

• **Database update.** The HTML page will send the whole data list to a JSP page (first, you must `JSON.stringify` the data list). The logic of that JSP page should be:
  o records without IDs (in data list) should be inserted into the database,
  o records not present (in the data list) should be deleted from the database (or you could keep a list of IDs to be deleted and send that as a separate ajax call).

**Submission:**

• Incorporate your challenge code in with the rest of your project.
• Include a blog about this challenge and a link to this code.
• Publish your project
• Submit a zip file into the "Client Side Challenge" BB assignment.