Programming the Browser

Programming interactive Webpages with
JavaScript and jQuery

Introduction

This is the second assignment in a set of related assignments that build on each other. This assignment assumes you have completed the previous assignment. First complete the first assignment before continuing on this assignment.

This assignment implements user management features such as creating new users, listing existing users, editing existing users, and removing users.

Learning Objectives

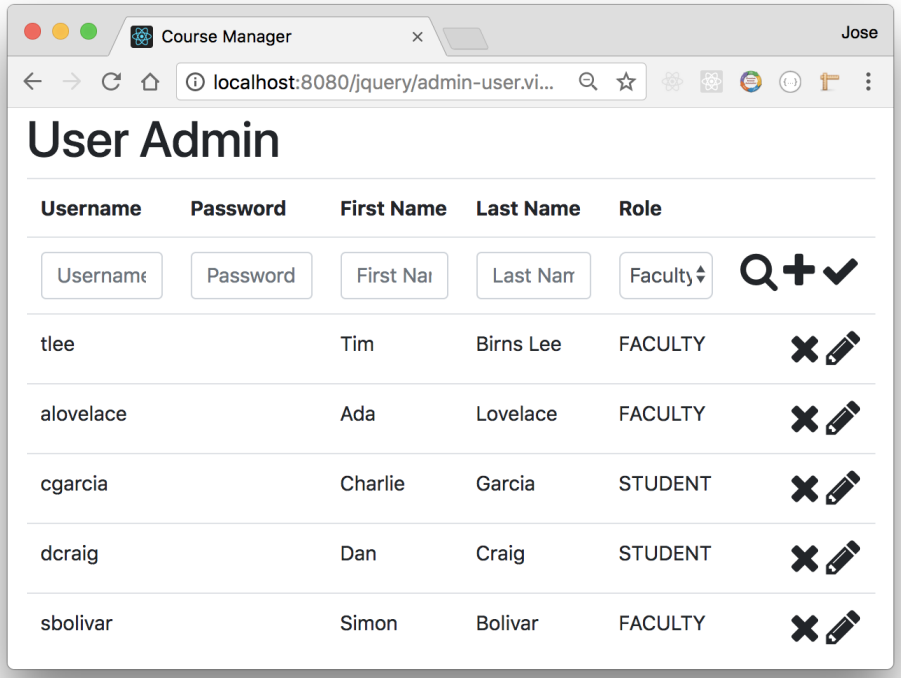
This assignment will introduce the student to the following concepts and skills:

- Creating webpages with the Hypertext Markup Language (HTML)
- Styling HTML web pages with the Bootstrap Cascading Style Sheet (CSS) library
- Programming web pages with the JavaScript programming language
- Dynamically manipulating a webpage's Document Object Model (DOM) with the jQuery JavaScript library
- Representing data using JavaScript Object Notation (JSON)
- Data modeling with the Unified Modeling Language (UML)
- Retrieving data through RESTful APIs

User admin requirements

The user admin page provides system administrators with the capability to create new users, retrieve all users, edit existing users, and delete users. The following wireframe suggests a user interface that provides all these use cases. You are free to improve on the design as long as you provide the minimum functionality described in this document.

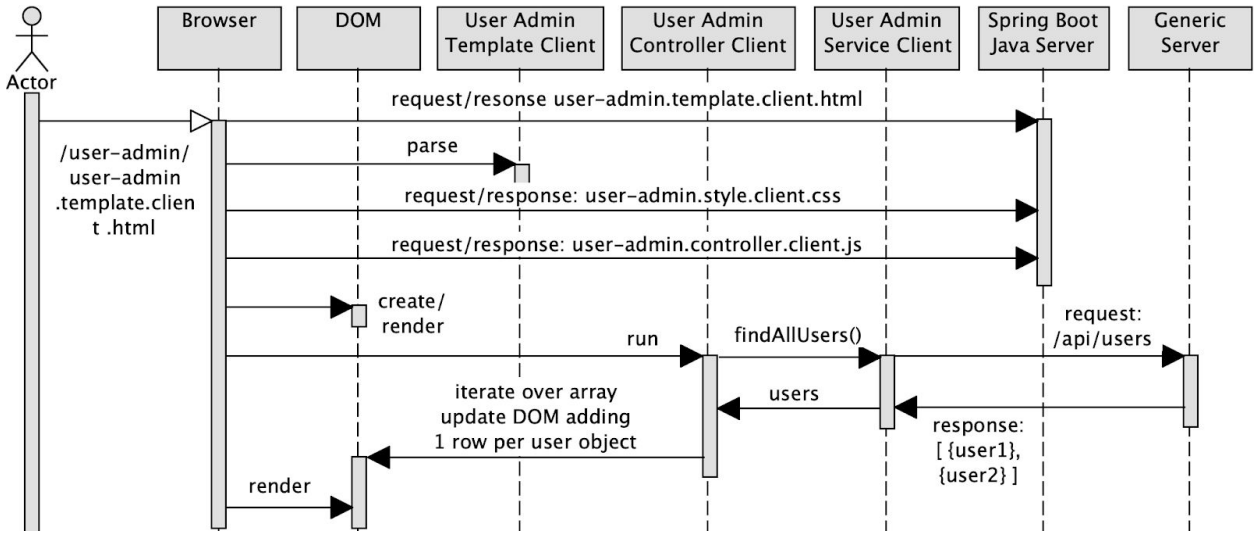
Push all code to the same github repository you used in previous assignments, and deploy your project to the same remote server you used in previous assignments.



User admin wireframe

The first row of the user admin web page displays column headings Username, Password, First Name, Last Name and Role. The second row has a form with placeholders. The last column of the second row has three icons to invoke the following actions: add new user and update an existing user. Administrators can create new users by filling the form and clicking on the add new user icon -- the plus sign. New users appear as a new row at the bottom of the table. When the webpage first loads, it renders all the users in the system, one user per row. The last column of each user row has two icons to invoke the following actions: remove user and edit user. Users can be removed by clicking on the remove user icon -- the x sign. To edit a user, admins click on the edit user icon -- the pencil icon -- and the user info appears in the top form. To update the selected user, the admin can edit the form and then click on the update user icon -- the checkbox icon.

Implement the user admin page using HTML, CSS, Bootstrap and jQuery. HTML (HyperText Markup Language) is a declarative language used to create and layout the content of Web pages. CSS (Cascading Style Sheets) is a declarative language used to style Web pages, that is, configure the look and feel of webpages. JavaScript is a programming language that allows interacting with and controlling a browser, particularly what the browser displays on the screen. jQuery is a popular JavaScript library used to dynamically manipulate the DOM easily. The DOM (Document Object Model) is the in-memory representation a browser creates when it parses an HTML document. The browser uses the DOM to represent and then render it as a webpage.



Design / Client Server Architecture

Each webpage will consist of two files: a template file, and a controller file. Template files are just HTML documents that provide an initial content and are meant to be manipulated by dynamically by JavaScript. Templates will render the user interface with which the user interacts. Controller files are just JavaScript files meant to work together with a related templates. In this assignment controller files will be using jQuery to dynamically interact with corresponding HTML template. Controllers are responsible for listening to user events and inputs, processing those events, manipulating a data model, and then rendering dynamic portions of the page. Additional JavaScript files implement services and models shared among several controller files. Service JavaScript files will encapsulate all data access to and from the server.

The user administration page will provide user administrators with the functionality of creating, retrieving, updating and removing users of the system. The following files implement the user administration page.

File	Description
`user-admin.template.client.html`	implements the user interface using HTML
`user-admin.controller.client.js`	implements the controller handling user events and rendering dynamic portions of the user admin page
`user.service.client.js`	implements the user service client encapsulating all data communication with the server

Notice the file naming convention. The naming follows the pattern

`{feature}.{role}.{tier}.{format}`

where the file parts mean the following

File part	Description	Example values
`feature`	describes a feature, component, or page	`user-admin, login, register, profile`

`role`	describes the role of the file in the overall architecture of the application	`model, service, template, controller, style, component`
`tier`	describes where in the multi tier application does the file execute	`client, server`
`format`	file extension describing the language the file is written in	`js, html, css, ts, sql, json, xml`

This naming convention allow developers to infer the purpose of a file by its name, and quickly orient themselves where in the application are they working on. All files must follow a naming convention agreed by the development team. Follow the above naming convention unless you have a better one. Check with your instructor if you prefer using a different naming convention.

User admin template

The user admin template renders the user interface for the user admin page using HTML and CSS as shown in the user admin wireframes. Implement the template in a file called `user-admin.template.client.html` in the `webapp/user-admin` folder. The following are code snippet samples that can be used as a starting point. Feel free to change the markup or provide an entirely better implementation

Loading relevant JavaScript libraries and stylesheets

The first thing to do is to load all relevant scripts and stylesheets. Use the link and script elements to load these dependencies. Here's an example:

webapp/user-admin/user-admin.template.client.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>User Admin</title>
<link rel="stylesheet" href=
  "https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"/>
<link  rel="stylesheet"  href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"
integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvKxVfELGroGkvsg+p"
crossorigin="anonymous"/>
<link href="user-admin.style.client.css" rel="stylesheet" />
<script                                  src="https://code.jquery.com/jquery-3.5.1.min.js"
integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
crossorigin="anonymous"></script>
<script src="../services/user.service.client.js"></script>
<script src="user-admin.controller.client.js"></script>
</head>
<body>
...
</body>
```

The `user.service.client.js`, `user-admin.controller.client.js`, and `user-admin.style.client.css` files are discussed later. The following sections describe the content of the `body` element.

Table headings

A table head element can render meta data, such as titles, for columns containing data of the same type. For instance, the following table head renders column headers for username, password, first and last name. The last column is left intentionally empty (` `) so that each row below it can render its own action links or buttons.

webapp/user-admin/user-admin.template.client.html

```
<table class="table">
  <thead>
    <tr>
      <th>Username</th>
      <th>Password</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Role</th>
      <th> </th>
    </tr>
```

Create/update user form

The second row of the table head can be used as a form with input fields for each of the properties of a new user, e.g., username, password, first and last name. The last column renders two action icons to update and create a user. Clicking on create user (plus icon) creates a new user instance and adds them to the bottom of the list. Clicking on update (check mark) updates the currently selected user and the corresponding row shows the updated fields. A user is selected for editing by clicking on edit (pencil icon) on the row of the user (described later in this document). Clicking on create or update clears the form.

webapp/user-admin/user-admin.template.client.html

```
<tr class="wbdv-form">
  <td><input id="usernameFld" class="form-control"
    placeholder="Username" /></td>
  <td><input id="passwordFld" class="form-control"
    placeholder="Password" /></td>
  <td><input id="firstNameFld" class="form-control"
    placeholder="First Name" /></td>
  <td><input id="lastNameFld" class="form-control"
    placeholder="Last Name" /></td>
  <td><select id="roleFld" class="form-control">
      <option value="FACULTY">Faculty</option>
      <option value="STUDENT">Student</option>
      <option value="ADMIN">Admin</option>
  </select></td>
  <td><span class="pull-right" style="white-space: nowrap">
```

```
      <i class="fa-2x fa fa-plus wbdv-create"></i>
      <i class="fa-2x fa fa-check wbdv-update"></i>
  </span></td>
</tr>
```

Notice the naming convention of adding `Fld` at the end of the input fields and prepending `wbdv` to our own classes to distinguish them from classes implemented by other libraries such as bootstrap or fontawesome. Make sure to use the IDs and classes as shown in the example snippet of code.

User template row

The list of users will be rendered in the table body as a collection of rows, one row per user. Each column of the row will render a different user property. An invisible row can be used as a template to be cloned and populate for every instance in a user array. Here's an example of what the table row template might look like.

webapp/user-admin/user-admin.template.client.html

```
  <tbody class="wbdv-tbody">
    <tr class="wbdv-template wbdv-user wbdv-hidden">
      <td class="wbdv-username">ada</td>
      <td> </td>
      <td class="wbdv-first-name">Ada</td>
      <td class="wbdv-last-name">Lovelace</td>
      <td class="wbdv-role">FACULTY</td>
      <td class="wbdv-actions">
        <span class="pull-right">
          <i class="fa-2x fa fa-times wbdv-remove"></i>
          <i class="fa-2x fa fa-pencil wbdv-edit"></i>
        </span>
      </td>
   </tr>
 </tbody>
</table>
```

Clicking on remove (X icon) removes the user from the list. Clicking on edit (pencil icon) selects the user as the currently selected user and copies their data into the form so the admin can update their information. Clicking on update (check mark) saves the changes permanently, updating the corresponding row for that user.

Commit your work into source control

As you complete a significant amount of work, and reach a stable state of your project, commit your work into source control. Add and commit the files created in this section. Use a commit command and message similar to the one below.

git commit command and message

```
git add .
git commit -m 'Created user admin template'
```

Do the same for every section that follows where a new file is created, or new functionality is added.

User admin styling

The user admin page should use the bootstrap css library and the fontawesome font library (or equivalent) to provide a consistent look and feel across the application. For instance, notice that the table stretches across the entire page as opposed to the default HTML table styling. Also notice that all content has consistent padding and margins, avoiding content to be flush with their containers. Bootstrap provides many code snippets that can be reused without having to reinvent the layout and styling. Feel free to make use of predefined code snippets, but avoid copying entire pages. Implement the user admin styling in a file called `user-admin.style.client.css` in the folder `webapp/user-admin`. Here's a snippet of code on how to load the CSS files from the HTML template file. Don't forget to commit your work.

webapp/user-admin/user-admin.template.client.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>User Admin</title>
<link rel="stylesheet" href=
  "https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"/>
<link  rel="stylesheet"  href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"
integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvKxVfELGroGkvsg+p"
crossorigin="anonymous"/>
<link href="user-admin.style.client.css" rel="stylesheet" />
<script
  src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="admin-user.service.client.js"></script>
<script src="admin-user.controller.client.js"></script>
</head>
```

User admin controller

Controllers are responsible for handling user generated input and events, and providing data for rendering the view. The user admin controller will handle events such as create user, find all users, find user by id, update user, and delete user. Implement the user admin controller in a file called `user-admin.controller.client.js` in the `webapp/user-admin` folder. The controller should get a reference to the form elements in the corresponding template, bind event handlers to create a new user, find, update and delete an existing user, and retrieve all users. Here's a template for the user admin controller. You'll need to implement the functions listed in the controller. Commit your work when done.

webapp/user-admin/user-admin.controller.client.js

```
(function () {
    var $usernameFld, $passwordFld;
    var $firstNameFld, $lastNameFld, $roleFld;
    var $removeBtn, $editBtn, $createBtn;
    var $userRowTemplate, $tbody;
    var userService = new AdminUserServiceClient();
```

```
    $(main);

    function main() { ... }
    function createUser() { ... }
    function deleteUser() { ... }
    function selectUser() { ... }
    function updateUser() { ... }
    function renderUsers(users) { ... }
    function findAllUsers() { ... } // optional - might not need this
    function findUserById() { ... } // optional - might not need this
})();
```

The user admin controller will have to implement the following event handlers.

webapp/user-admin/user-admin.controller.client.js

Function	Description
`main()`	executes on document load, when the browser is done parsing the html page and the dom is ready. Retrieved the dom elements needed later in the controller such as the form elements, action icons, and templates. Binds action icons, such as create, update, select, and delete, to respective event handlers
`createUser()`	handles create user event when user clicks on plus icon. Reads from the form elements and creates a user object. Uses the user service createUser() function to create the new user. Updates the list of users on server response
`findAllUsers()`	called whenever the list of users needs to be refreshed. Uses user service findAllUsers() to retrieve all the users and passes response to renderUsers
`findUserById()`	called whenever a particular user needs to be retrieved by their id, as in when a user is selected for editing. Reads the user id from the icon id attribute. Uses user service findUserById() to retrieve user and then updates the form on server response
`updateUser()`	handles update user event when user clicks on check mark icon. Reads the user id from the icon id attribute. Reads new user values form the form, creates a user object and then uses user service updateUser() to send the new user data to server. Updates user list on server response
`deleteUser()`	handles delete user event when user clicks the cross icon. Reads the user id from the icon id attribute. Uses user service deleteUser() to send a delete request to the server. Updates user list on server response
`renderUsers()`	accepts an array of user instances, clears the current content of the table body, iterates over the array of users, clones a table row template for each user instance, populates the table row with the user object properties, adds the table row to the table body

User admin service client

The user admin service implements all data acces. Implement the user admin service in a file called `user.service.client.js` in the `webapp/services` folder. Here's a template for the user service. **Last reminder: commit your work when done.**

webapp/services/user.service.client.js

```
function AdminUserServiceClient() {
    this.createUser = createUser;
    this.findAllUsers = findAllUsers;
    this.findUserById = findUserById;
    this.deleteUser = deleteUser;
    this.updateUser = updateUser;
    this.url = 'https://wbdv-generic-server.herokuapp.com/api/YOURNEUID/users';
    var self = this;
    function createUser(user) { ... }
    function findAllUsers() { ... }
    function findUserById(userId) { ... }
    function updateUser(userId, user) { ... }
    function deleteUser(userId) { ... }
}
```

The user service should implement the following functions to interact with the user web service.

Function	Description
`createUser(user)`	accepts a user object and adds it to a collection of users
`findAllUsers()`	retrieves all users as an array of JSON objects
`findUserById(id)`	retrieves a single user object whose id is equal to the id parameter
`updateUser(id, user)`	accepts a user id and user object with new property values for the user with user id. Finds the user whose id matches he id parameter and updates it with the values in the user parameter
`deleteUser(id)`	removes the user whose id matches the id parameter

Use the generic server to send and retrieve data

Using jQuery's AJAX or fetch command, send and retrieve user information to the generic server discussed in class with the following URL patterns. The generic server can keep track of information of any number of users. Use your NEUID for {neuid} to distinguish your data from other students. Use HTTP POST to create data, GET to retrieve data, PUT to update data, and DELETE to remove data as described below. The URL to interact with the generic server is

https://wbdv-generic-server.herokuapp.com/api/YOURNEUID/users

Where YOURNEUID is your student ID.

Method	URL Pattern	Description
`POST`	`/api/{neuid}/users`	accepts a user object in HTTP BODY and adds it to a collection of users in the server

GET	/api/{neuid}/users	returns the server collection of all users as an array of JSON objects
GET	/api/{neuid}/users/{id}	retrieves a single user object whose id is equal to the id parameter
PUT	/api/{neuid}/users/{id}	accepts a user id and user object in the HTTP BODY with new property values for the user with user id. Finds the user whose id matches the id parameter and updates it with the values in the user parameter
DELETE	/api/{neuid}/users/{id}	removes the user whose id matches the id parameter

Deliverables

As a deliverable, check in all your work into a github source control repository. Deploy your project to a remote public server on Heroku or AWS. Submit the URL of the repository and the remote server in blackboard. Tag the commit you want graded as assignment2. TAs will clone your repository and grade the tagged commit.