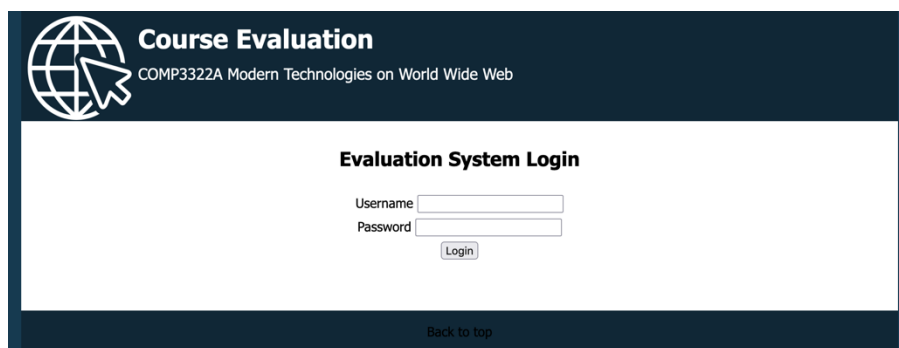# COMP3322A Modern Technologies on World Wide Web
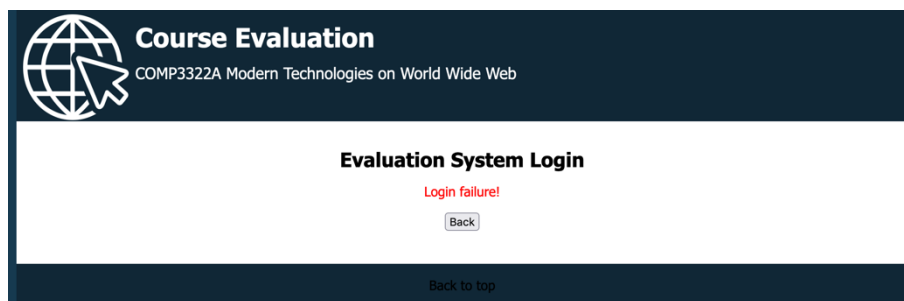## Lab 4: Node.js basics, AJAX, MongoDB
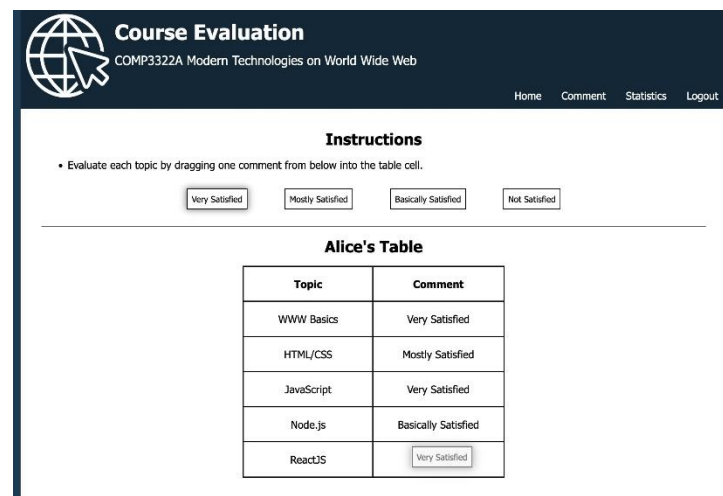
Overview

In this lab exercise, we will implement a simple course evaluation system, in which the client side uses AJAX to send requests, and the server side uses MongoDB to store data and Express.js to respond. A user first logs in with a valid username, and then the page will show an evaluation table. The user can evaluate topics of the course by dragging a pre-prepared comment to the corresponding table cell. Please refer to the following screenshots:

Login page

Redirect to a login failure page if the username and password are invalid.

Figure 1. Login with session management.

After login, the evaluation page will be displayed, showing instructions, four prepared comments and an evaluation table with the title showing the login user's name. The navigation bar will display a logout link (click and go back to the login page).

One can drag one comment among the four into the "Comment" column of a topic. Updates will be saved in the server side and will be reserved when refreshing the page or logout and then log in the system again.

Figure 2. Course Evaluation

Note: Due to compatibility of difference browsers, the same CSS code may lead to different display; we will check the lab in the <u>Chrome</u> browser. Therefore, you are recommended to use the <u>Chrome</u> browser to develop/test your page as well.

Lab Exercise

Part 1. Preparation

Step 1. Download the code templates from Moodle

Download "**lab4_materials.zip**" from HKU Moodle, and extract it to a folder. In this folder, you will find 3 JavaScript files ("**generate_db.js**", "**app.js**", "**index.js**"), 2 CSS files ("**main.css**" and "**basics.css**"), 3 HTML files ("**login.html**", "**comment.html**", "**fail.html**"), and 1 Image ("**logo.png**"). In this lab, we will ONLY edit "**comment.html**", "**index.js**" and "**app.js**", and keep other files unchanged. The HTML and JavaScript files can be edited using any code editors or IDEs.

Step 2. Create an Express app skeleton

Follow steps 1 to 3 in the handout "setup_nodejs_runtime_and_example_1.pdf" to create an Express app. Move files extracted from "**lab4_materials.zip**" to the corresponding subdirectory: (1) move "**index.js**" to "public/javascripts/"; (2) overwrite the original "**app.js**" in the Express app directory with the "**app.js**" we provided; (3) move "**main.css**" and "**basics.css**" to "public/stylesheets/"; (4) move "**login.html**", "**comment.html**", "**fail.html**" to "public/"; (5) move "**logo.png**" to "public/images/". If you check out contents of the HTML files, you will find that they are linking to the respective CSS, JavaScript and Image files. Besides, "**generate_db.js**" is an auxiliary JavaScript file to facilitate easier database initialization, which you will use next (you can keep this file anywhere you can find).

Step 3. Insert documents into MongoDB

Follow steps 1-3 in Example 6 of the handout "AJAX_JSON_MongoDB_setup_and_examples.pdf" to install MongoDB (if MongoDB is not yet installed on your computer), create a "data" folder in your Express app directory, and start the MongoDB server using the "data" directory of your project.

Launch another terminal, switch to the directory where mongodb is installed, and execute the following command:

```
./bin/mongo YourPath/generate_db.js
```

Make sure you replace "**YourPath**" by the actual path on your computer where you keep the "generate_db.js" that we provided.

This command runs the code in "generate_db.js". If you check out the content of "generate_db.js", you will find out that it creates a database "lab4-db" in the database server and inserts a few user documents into a **userList** collection in the database. Each document in the **userList** collection contains the following key-value pairs:

- _id: The unique ID of the document, which you do not need to include when inserting a document, and MongoDB server will add automatically into each inserted document. You can check _id of all inserted documents using db.userList.find() in the interactive shell (refer to step 4 of Example 6 in the handout "AJAX_JSON_MongoDB_setup_and_examples.pdf").

- username: The name of the user.
- password: The password of the user.
- comment: An object containing key-value pairs, where a *key* is a course topic to evaluate, and the *value* is a comment given by the user (initialized as empty strings).

Part 2. Implement login validation.

Step 4. Process HTTP GET request for http://127.0.0.1:8081/

Open "**app.js**". Complete the callback function of route **app.get('/', (req, res) =>{...})**: if the session variable loginName has been set, send "**comment.html**" to the client; otherwise, send "**login.html**" to the client.

After this step, run the Express app using "node app.js", and visit http//127.0.0.1:8081/ to check the page out. The page should show up like the following:


Note: Everytime after you have modified **app.js**, you need to re-launch the server program for testing.

**Step 5.** Validate the user login and respond.

In "**login.html**", find the <section> element. We can see a <div> container with a <h2> heading which reads "Evaluation Sysetem Login". Right under the heading, we have a **<form>** element with attributes: (1) **action** of "http://127.0.0.1:8081**/login_post**", and (2) **method** of "**POST**". Inside the form, we have created 3 <input> elements for username, password and submit.

In "**app.js**", complete the callback function in route **app.post('/login_post', express.urlencoded({ extended: true }), (req, res) => {...})**, that handles the HTTP POST request sent by the client when the login form is submitted. In the callback function, use **collection.find({username: name})** with query **{username: name}** to find the user document whose username equals to the name carried in the request body. Check whether the password carried in the request body matches the record stored in the found document. If the password matches, store the name into the session variable "**loginName**" and send "**comment.html**" back to the client; otherwise, send "**fail.html**" to the client.

After this step, when you have successfully logged in, you will see a page like:
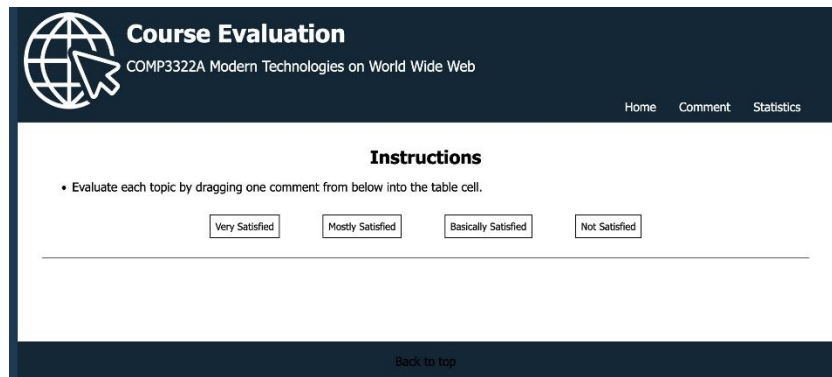
If your login fails, you will see a login failure page as follows:
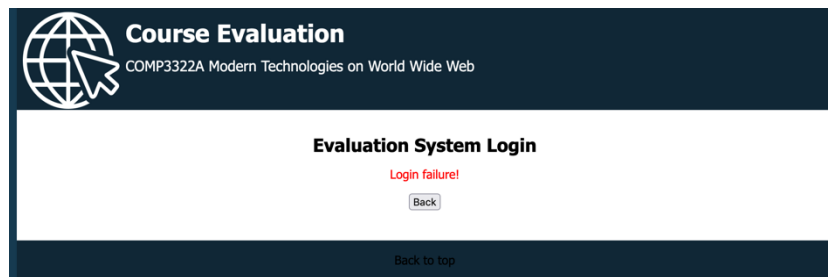
Part 3. Display table and implement course evaluation.

### Step 6. Implement the logout link and table title.

In "**comment.html**", we can see that the function **init()** is called when the page is loaded.

6.1. Open "**index.js**". In the **init()** function, use **XMLHttpRequest** to send a HTTP **GET** request for "**/get_user**". The server will process the request and respond with the user name in a string format (to be implement in **app.js**). Upon successfully receiving server response (*readystate==4* and *status==200*), get the html element with id "**login_user**", and set its **innerHTML** value to "*username*'s Table" (*username* should be replaced by the actual user name received from the server). Then create a <a> element and set its "href" attribute to "/logout" and innerHTML to "Logout". Append this hyperlink as a child to the node with id "**nav_out**".

6.2. In "**app.js**", complete the callback function of route **app.get('/get_user', (req, res) => {…})**, which sends the session variable **loginName**'s value to the client.

6.3. In "**app.js**", complete the callback function of route **app.get('/logout', (req, res) => {…})**, which resets the loginName session variable to *null* and redirects the client to '**/**'.

After this step, the navigation bar is updated and the table title can be displayed, as follows:
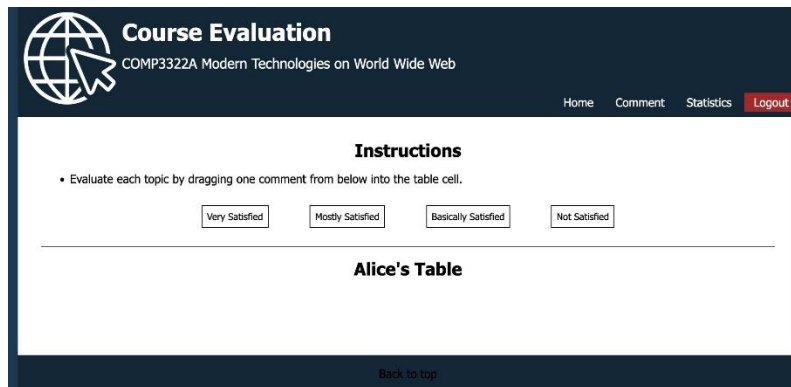
Step 7. Show course comments in a table.

7.1. In the client-side script "**index.js**", implement the **displayTable()** function to request from the server information of all topics and display them in a table, as follows: use **XMLHttpRequest** to send a HTTP **GET** request for "**/get_table**"; upon receiving the server response, get the table element in "**comment.html**" with id "**evaluate_table**" and set its **innerHTML** to the received *responseText* from the server (the server will render the table rows in HTML representation and send it back to the client).

7.2. In the server-side script "**app.js**", complete the callback function of route **app.get('/get_table', (req, res) => {…})**, by creating the HTML contents for displaying the rows in the table. We have provided the code to create the header row with 2 **<th>** elements and they are stored into the string "response". Use **collection.find({username: req.session.loginName})** to find the document of the logged-in user in the userList collection. You should create a row for each topic that is stored in the 'comment' field of the found document and use 2 **<td>** elements to display the 'topic' and 'comment', respectively. [Hint: you may find Object.keys (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys) useful to retrieve keys in a JavaScript object.] Make the **<td>** elements for the 'comment' column **droppable** by adding additional attributes to each **<td>** element: **ondrop="drop(event)"** and **ondragover**="**allowDrop(event)**". The **allowDrop(event)** function has been given in "**index.js**". We will implement the **drop(event)** function later. All HTML elements should be concatenated to the "response" string. Send the "response" string back to the client if the database operation is successful; otherwise, send the error message back.

(For reference of drag-and-drop, see page 24 of *7_HTML_withScript_COMP3322A_s2022.pdf*).

After finishing this step, you can display a table with 5 topics, and the comment columns haven't been filled.

Step 8. Course Evaluation.

8.1. In "**comment.html**", we can find a <**div**> element of id "comments" with 4 **<p>** tags showing prepared comments. To allow each comment to be draggable, add two attributes **draggable="true"** and **ondragstart="drag(event)"** in each tag. The function **drag(event)** is provided in "**index.js**".

8.2. In "**index.js**", complete the **drop()** function. Set the innerHTML of the target element, which the dragged element is dropped into, to the innerHTML of the dragged element, so that the dragged comment can be filled into the respective table cell. Obtain the topic that the comment is added to by getting the innerHTML of the sibling node of the target element. Send a HTTP **POST** request for **"comment_post"**. Set "**application/x-www-form-urlencoded**" as a request header and put the topic and comment in the request body in the format of "topic=xx&comment=yy" (replace xx by the topic, replace yy by the comment). Upon successfully receiving server response (a message string in *responseText*), **alert** this message.

8.3. In "**app.js**", complete the callback function of route **app.post('/comment_post', express.urlencoded({ extended: true}), (req, res) => {…})**. Use **collection.find({username: req.session.loginName})** to find the document of the logged-in user in the userList collection, retrieve its comments field and update the comments field according to the topic and comment carried in the request body, using **collection.update({username: req.session.loginName}, {$set: {comments: new_comments}})** where new_comments is the updated comments object (see the usage of collection.update at https://automattic.github.io/monk/docs/GETTING_STARTED.html). If the database operation is successful, send the string "Successfully commented!" to the client; otherwise, send the error in the response.

After this step, you can try dragging one prepared comment, and when you drop it into a table cell (in the "comment" column), you are able to see an alert message as follows. The updated comment will not be cleared when you refresh the page or log out and then log in the system again.

Congratulations! Now you have finished Lab 4. You should test the pages and the final results should look similar to the screenshots at the beginning of this document.

Submission:

Please finish this lab exercise before 23:59 Sunday October 30, 2022. Please compress the entire app folder (i.e., the folder in which you create the express app) into a .zip file and submit it on Moodle.

(1) Login Moodle.
(2) Find "Labs" section and click "Lab 4".
(3) Click "Add submission", browse your .zip file and save it. Done.
(4) You will receive an automatic confirmation email, if the submission was successful.
(5) You can "Edit submission" to your already submitted file, but ONLY before the deadline.