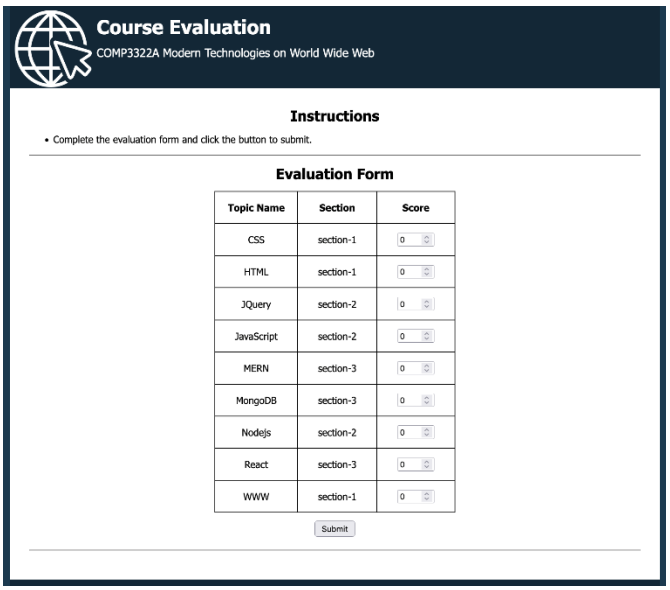
COMP3322A Modern Technologies on World Wide Web

Lab 5: JSON, MongoDB, JQuery

Overview

In this lab exercise, we will implement a simple course evaluation web app, in which the client side uses jQuery AJAX to send requests, and the server side uses MongoDB to store data and sends responses in JSON. The web page will show a table containing scores given to each topic. One can enter a score to each topic and click the submit button to save the input data to the server side. Please refer to the following screenshots:



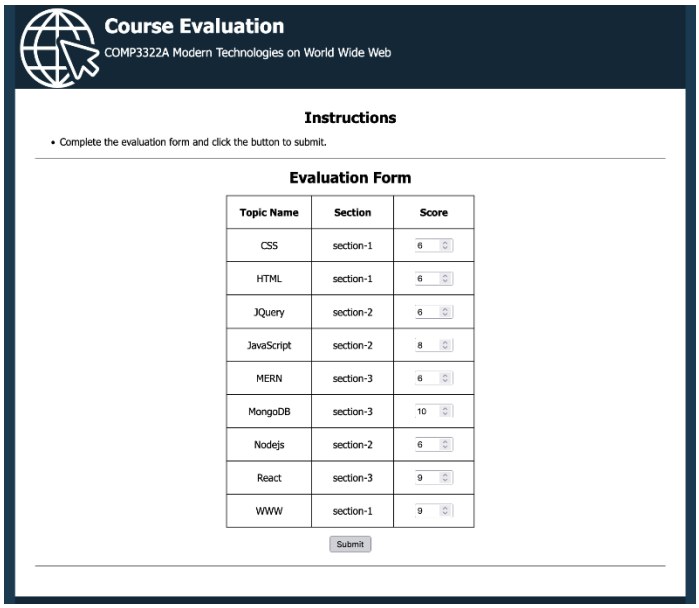
In the table on the page, rows are sorted in alphabetical order of the topic name. Each topic is initialized with a score of 0.

Evaluation Form

Topic Name	Section	Score
CSS	section-1	2
HTML	section-1	3
JQuery	section-2	10

You can click the up/down arrows of input boxes or enter a number (within the range of 0 to 10) to score each topic.

Figure 1. Display the Table



When you have entered your scores, you can click the submit button to save your scores into the server-side database.

Figure 2. Submit Form

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

Lab Exercise

Part 1. Preparation

Step 1. Download the code templates from Moodle

Download "**lab5_materials.zip**" from HKU Moodle, and extract it to a folder. In this folder, you will find 3 JavaScript files ("**externalJS.js**", "**app.js**", "**generate_db.js**"), 2 CSS file ("**basics.css**", "**main.css**"), 1 HTML file ("**form.html**"), and 1 Image ("**logo.png**"). In this lab, we will ONLY edit "**externalJS.js**" and "**app.js**", and keep other files unchanged.

Step 2. Create an Express app skeleton

Follow step 1 to step 3 in the handout "setup_nodejs_runtime_and_example_1.pdf" to create an Express app. Move files extracted from "**lab5_materials.zip**" to the corresponding subdirectory: (1) move "**externalJS.js**" to "public/javascripts/" and "**generate_db.js**" to the Express app directory; (2) overwrite the original "**app.js**" in the Express app directory with the "**app.js**" we provided; (3) move "**basics.css**", "**main.css**" to "public/stylesheets/"; (4) move **"form.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 jQuery library, respective CSS, JavaScript and Image files.

Step 3. Insert documents into MongoDB

Follow steps in Example 6 of the handout "AJAX_JSON_MongoDB_setup_and_examples.pdf" to run a MongoDB server. Launch another terminal, switch to the directory where mongodb is installed, and execute the following command:

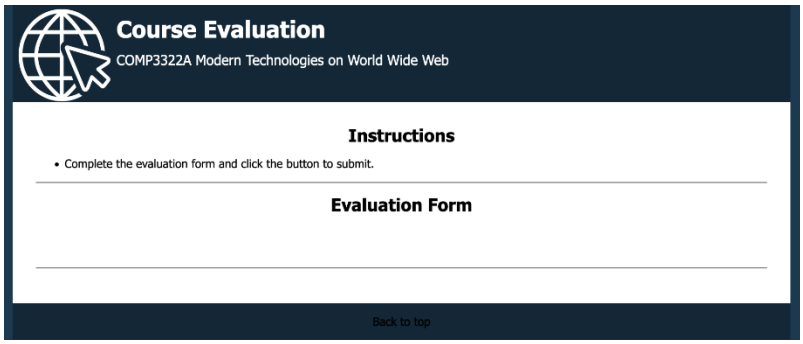
```
./bin/mongo YourPath/generate_db.js
```

Make sure you replace "**YourPath**" by the actual path on your computer where you keep the "generate_db.js" that we provided. The script will create a database "lab5-db" in the database server and insert a few documents into a **topicList** collection in the database. Each document in the **topicList** 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.topicList.find()** in the interactive shell.
- **name**: the name of the topic.
- **section**: The name of the section that the topic belongs to.
- **score**: The evaluation score on the topic.

For this line of code 'section':`section-${id}` in "generate_db.js", we are using JavaScript's template literal with the backtick ` in the `section-${id}` part, for easy generation of a JavaScript string that involves some variable's value (e.g., id in this example). Read more about template literals at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals, which you can exploit for easier generation of strings concatenating some variable values.

In "app.js", we have provided some basic code. After launching the Express server and accessing http://127.0.0.1:8081/, you will see a page as follows:



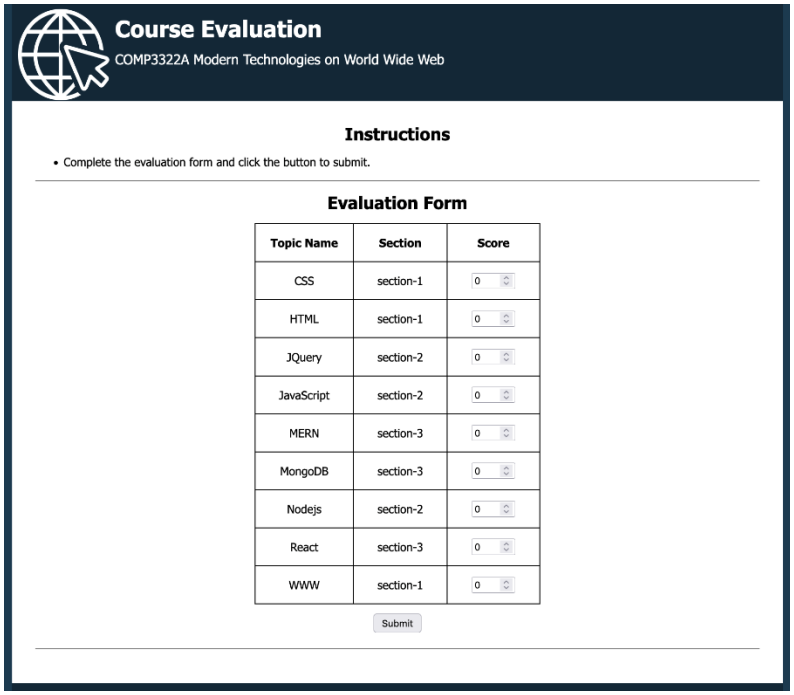
Part 2. Implement score table functionalities

Step 4. Show the evaluation table and submit button.

4.1. Open "**app.js**", complete the callback function in **app.get('/get_form', (req, res) => {…})** as follows. Use collection**.find()** to find all documents in the **topicList** collection and use.**sort()** to sort them in alphabetical order of the topic name (see https://www.tutorialspoint.com/mongodb/mongodb_sort_record.htm, https://www.mongodb.com/docs/upcoming/reference/method/cursor.sort/ and https://www.mongodb.com/docs/manual/reference/operator/aggregation/sort/). In the returned promise's callback **then((docs) => {…})**, use **res.json()** to send **docs** (an array of JSON objects) that contains the found topic documents back to the client.

4.2. Open the client-side script "**externalJS.js**". We see that the function **showEvalForm()** is called when the page is loaded, which will display all topic rows in a table and render the submit button. Complete the **showEvalForm()** function as follows. First, request from the server all topic documents and display them in a table format. Different from lab4, here we use **$.getJSON()** with url "**/get_form**" to send a HTTP **GET** request to the server side. In the callback function of **$.getJSON()**, store the received array of topic documents in the given global variable *record_list* (which will be used in a later step). Then use **$.each()** to iterate through these documents: for each document, create the HTML representation of a table row with three <td> elements - the first two <td> elements enclose the "**name**" and "**section**" fields of the document and the third <td> element contains an <input> element with id="name" field of the document, class="score", type="number", value="score" field of the document, min=0 and max=10 (indicating the score range). The row representations should all be concatenated into a string *table_content*; use **$('#evaluate_table').html()** to set *table_content* as the HTML content of the table with id "evaluate_table". Further, create a button element and use **$("#div_button").append()** to append the button to the of id "div_button"; bind **submitForm** as the handler function of click event on the button**.** We will implement the **submitForm** function in a later step.

After this step, you can see all topic scores shown in a table. The page should show up like the following:

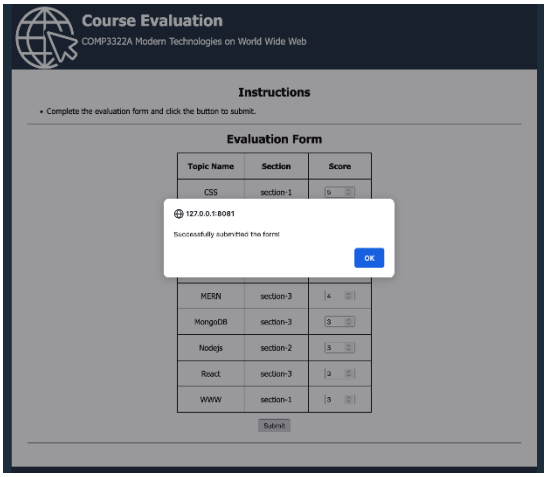


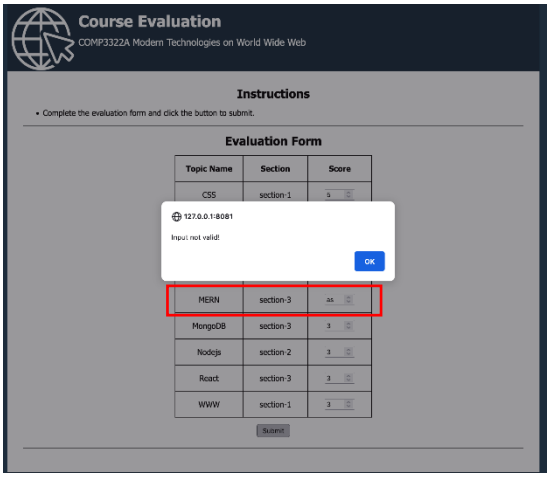
Step 5. Update scores to database

5.1. Open "app.js", complete the callback function in **app.post('/update_score', express.urlencoded({ extended: true }), (req, res) => {…})** as follows. Use collection**.update()** with query **{'name': req.body.name}** and update operation **{$set:{'score':parseInt(req.body.score)}}** to set the score field of the respective document in the **topicList** collection (whose "**name**" matches the query). In the callback function of collection**.update()**, if the error obtained by the update method is *null*, send the JSON object **{msg: "Successfully submitted the form!"}** to the client; otherwise, send **{msg: error}** in the response.

5.2. Open "**externalJS.js**", implement the function **submitForm(event)**, which is invoked when the submit button is clicked. Use **event.preventDefault()** to cancel any default behavior of the button click. Then check whether all inputs in the number input boxes are valid numbers (we have provided the function **isInputValid()** for this checking). If not all valid, alert "*Input not valid!*"**.** If all valid, do the following: Update the score field of each topic document in the array of topic documents stored in the global variable *record_list* with **$(`#${record_list[i].name}`).val()** (this selector selects the input element whose id equals the name of the topic document). For each topic document in *record_list*, use **$.post** to send a HTTP POST request with url **"/update_score"** and data "**record_list[i]**"; upon receiving a response from the server (which is a JSON object containing a "**msg**" key), alert the value of "**msg**".

After this step, after modifying the scores and clicking the submit button, the updates will be saved into the database.





Congratulations! Now you have finished Lab 5. 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 Thursday November 10, 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 5”.
(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.