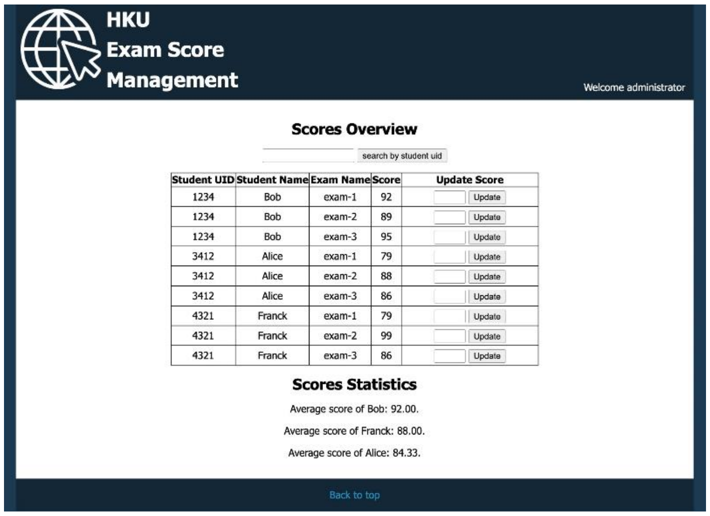
COMP3322B Modern Technologies on World Wide Web

Lab 5: JSON, MongoDB

Overview

In this lab exercise, we will implement a simple score management 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. The page will show a table containing records of students' scores in different examinations. You can filter records by student's uid and update the score for a record. The average score of each student in all exams will be calculated and shown underneath the score table. Please refer to the following screenshots:



Scores Overview

Student UID	Student Name	Exam Name	Score	Update Score
1234	Bob	exam-1	92	Update
1234	Bob	exam-2	89	Update
1234	Bob	exam-3	95	Update
3412	Alice	exam-1	79	Update
3412	Alice	exam-2	88	Update
3412	Alice	exam-3	86	Update
4321	Franck	exam-1	79	Update
4321	Franck	exam-2	99	Update
4321	Franck	exam-3	86	Update

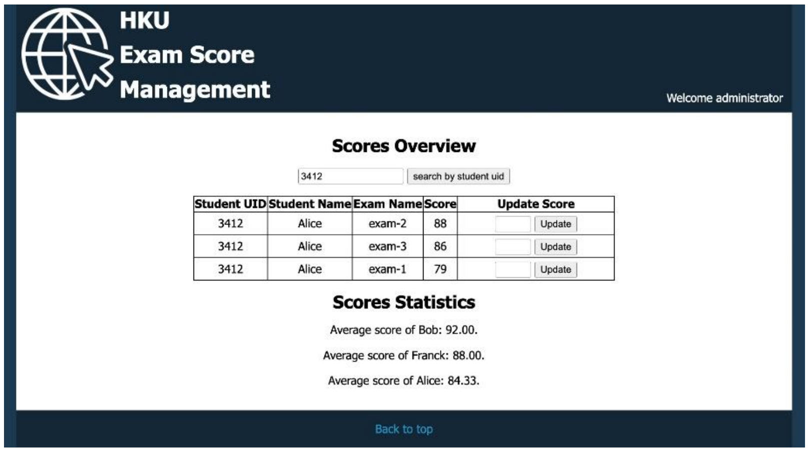
Scores Statistics

Average score of Bob: 92.00.

Average score of Franck: 88.00.

Average score of Alice: 84.33.

Upon initial page load, you will see a score table (in ascending order of student UID), and score statistics of each student's average score (in descending order of average scores).



Scores Overview

3412 | search by student uid

Student UID	Student Name	Exam Name	Score	Update Score
3412	Alice	exam-2	88	Update
3412	Alice	exam-3	86	Update
3412	Alice	exam-1	79	Update

Scores Statistics

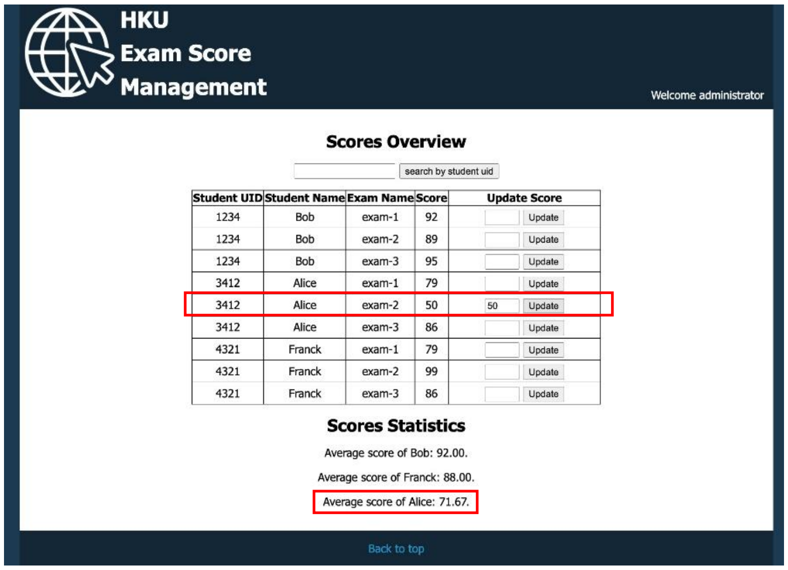
Average score of Bob: 92.00.

Average score of Franck: 88.00.

Average score of Alice: 84.33.

You can type a student's uid in the input box to display only the scores of this student (in descending order of scores).

Figure 1. Show Score Table and Statistics



Scores Overview

search by student uid

Student UID	Student Name	Exam Name	Score	Update Score
1234	Bob	exam-1	92	Update
1234	Bob	exam-2	89	Update
1234	Bob	exam-3	95	Update
3412	Alice	exam-1	79	Update
3412	Alice	exam-2	50	50 Update
3412	Alice	exam-3	86	Update
4321	Franck	exam-1	79	Update
4321	Franck	exam-2	99	Update
4321	Franck	exam-3	86	Update

Scores Statistics

Average score of Bob: 92.00.

Average score of Franck: 88.00.

Average score of Alice: 71.67.

When updating a score by tying the new score into corresponding input box in the "Update Score" column and clicking the "Update" button, the "Score" and the average score of the student displayed will be updated accordingly (red boxes are for illustration only and do not belong to the page). For example, after we update the score of Alice in exam-2 from 88 to 50, Alice's average score changes from 84.33 to 71.67.

Figure 2. Update Score

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 (“**index.js**”, “**app.js**”, “**generate_db.js**”), 1 CSS file (“**main.css**”), 1 HTML file (“**scores.html**”) and 1 Image (“**logo.png**”). In this lab, we will ONLY edit “**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 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) overwrite the original “**app.js**” in the Express app directory with the “**app.js**” we provided; (2) move “**index.js**” into “public/javascripts/”; (3) move “**main.css**” to “public/stylesheets/”; (4) move **“scores.html**” to “public/”; (5) move “**logo.png**” to “public/images/”. If you check out the content of the HTML file, you will find that it is linking to “**index.js**”, the CSS and Image files. Besides, “**generate_db.js**” is an auxiliary JavaScript file to facilitate easier database data initialization, which you will use next (you can keep the 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
```

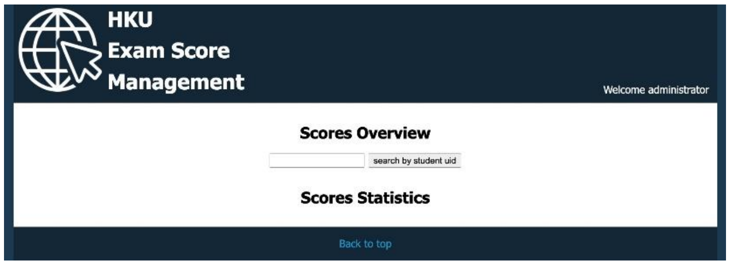
Make sure you replace “**YourPath**” by the actual path on your computer where you keep the “generate_db.js” that we provided.

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 “lab5-db” in the database server and inserts a few student documents into a **stuList** collection in the database. Each document in the **stuList** 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.stuList.find()** in the interactive shell (refer to step 4 of Example 6 in the handout “AJAX_JSON_MongoDB_setup_and_examples.pdf”).

- **uid**: The university ID number of the student
- **name**: The name of the student
- **exam**: The exam that the student has taken
- **score**: The score achieved by the student in the exam

We have linked the Express app with MongoDB in "app.js" provided. After launching the Express app (using "node app.js" in your Express app directory), access http://127.0.0.1:8081/ and you will see a page without any scores information as follows:



Part 2. Implement scores overview and statistics

Step 4. Show scores information in a table and filter them by UID.

In "**scores.html**", we see that the function **findAllDocs()** is called when the page is loaded, which requests from the server information of all score records and display them in a score table. Onclick of the button "search by student uid" is associated with event handler **findDocsByUID(),** which retrieves and displays score records of the uid entered in the search box.

4.1. Open the client-side script "**index.js**". Implement **findAllDocs()** function as follows: it uses **XMLHttpRequest** to send an HTTP **GET** request for "**get_scores**"; upon receiving the server response, get the table element in "**scores.html**" of id "**scores_table**" and set its **innerHTML** to the received *responseText* from the server (the server will find all documents from the **stuList** collection in the database, render the score information in HTML representations and send it back to the client). At the end of the function, we call **calStatistics()** function to calculate the average score of each student and display them on the page (we will implement it later).

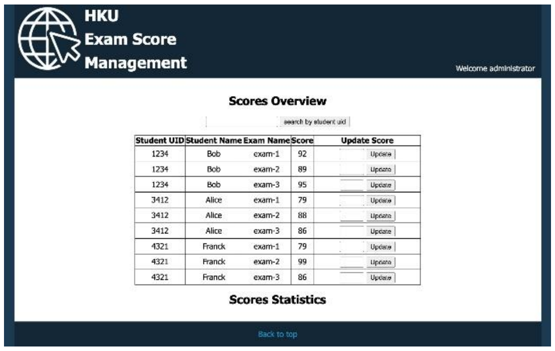
4.2. Implement **findDocsByUID()** in "**index.js**" as follows: use **XMLHttpRequest** to send an HTTP **GET** request for "**get_scores**" with a query parameter *find,* whose value should be the value entered in the input textbox of id "**find_input**"; upon receiving server response, get the table element in "**scores.html**" of id "**scores_table**" and set its **innerHTML** to the received *responseText* from the server.

4.3. Open the server-side script "**app.js**". Complete the callback function in **app.get('/get_scores', (req, res) =>{…})**: (1) if the *find* parameter is *null* or an empty string "" (which means the request is sent from **findAllDocs()** to find all documents in the **stuList** collection, or the client clicks the "search by student uid" button with empty content in the input textbox), use collection**.find({},**

{sort: {uid: 1}}) with the sort option to find all documents in the **stuList** collection and sort the retrieved documents in ascending order of uid (read more about the find and the sort option at https://automattic.github.io/monk/docs/collection/find.html and http://mongodb.github.io/node-mongodb-native/3.2/api/Collection.html#find); (2) otherwise, use collection.**find({uid: find_value}, {sort: {score: -1}})** with query **{uid: find_value}** and sort option **{sort: {score:-1}}** to find documents with the same uid as **find_value** (the value of query parameter *find*) and sort retrieved documents in descending order of **score**.

The find operation on database is asynchronous and returns a promise. In the promise's callback **then((docs) => {…})**, call **renderHTML()** to convert the retrieved JSON array *docs* into an HTML string and send the HTML string back to the client. We have provided the code of **renderHTML()**, which creates the header row of a table with five **<th>** elements and creates one table row for each document with five **<td>** elements. In each row, for the column "Update Score", it creates an <input> element with id "new_score_${doc.uid}_${doc.exam}" and a <button> element with onclick event handler **updateScore('${doc.uid}', '${doc.exam}')** (we will implement the updateScore function later). To understand how we use **${variable}** for variable substitution in JavaScript strings, read more about JavaScript template literals using the back-tics (` `) and string interpolation at https://www.w3schools.com/js/js_string_templates.asp.

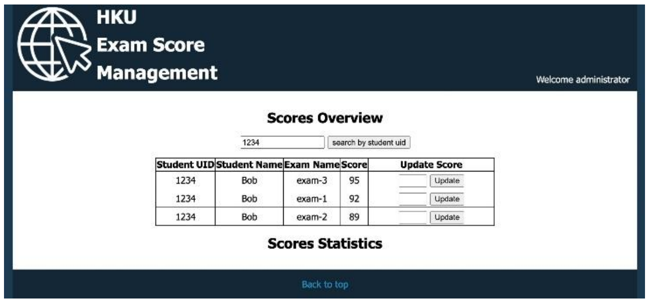
After this step, you are able to see all scores shown in a table, and you can filter them by student's uid. The page should show up like the following (the score statistics part has not been implmented yet):



Scores Overview

Student UID	Student Name	Exam Name	Score	Update Score
1234	Bob	exam-1	92	Update
1234	Bob	exam-2	89	Update
1234	Bob	exam-3	95	Update
3412	Alice	exam-1	79	Update
3412	Alice	exam-2	88	Update
3412	Alice	exam-3	86	Update
4321	Franck	exam-1	79	Update
4321	Franck	exam-2	99	Update
4321	Franck	exam-3	86	Update

Scores Statistics



Scores Overview

Student UID	Student Name	Exam Name	Score	Update Score
1234	Bob	exam-3	95	Update
1234	Bob	exam-1	92	Update
1234	Bob	exam-2	89	Update

Scores Statistics

Step 5. Show scores statistics

5.1. In "**index.js**", implement the function **calStatistics()**. We have provided the code for first clearing the element container of id "**statistics**". Then use **XMLHttpRequest** to send an HTTP **GET** request for "**get_statistics**". Upon receiving the server response, which is a JSON string, use **JSON.parse**() to convert *responseText* into a JSON array. Each entry in the array is a JSON object containing two key-value pairs (defined in the server-side code): (1) "**_id**": the student name; (2) "**avg_score**": the average score of the student in all exams. Create a <p> element for each entry, with its innerHTML set to "Average score of $name: $score", where $name and $score are the values of "**_id**" and "**avg_score**", respectively. You can show only two decimal places of each average score in the page (use **.toFixed(2)**: https://www.w3schools.com/jsref/jsref_tofixed.asp). Then append these <p> elements in the div element of id "**statistics**".

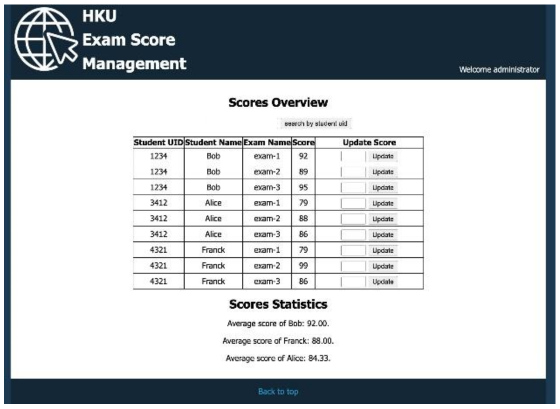
5.2. In "**app.js**", complete the callback function in **app.get('/get_statistics', (req, res) =>{…}).** In the callback function, use collection**.aggregate()** to calculate aggregate values of data in the **stuList** collection (learn about the **aggregate** method at https://automattic.github.io/monk/docs/collection/aggregate.html), with its *pipeline* array parameter (which specifies a sequence of data aggregation operations) defined as follows:

```
[
    {$group:
      {_id:"$name",
      avg_score:{$avg:"$score"}
      }
    },
    {$sort: {avg_score: -1}}
 ]
```

Here **{$group: {_id:"$name", avg_score:{$avg:"$score"}}}** operation specifies that documents should be grouped by their "**name**" field, and the average of the "**score**" field of documents in each group will be calculated and the result will be stored with the key "**avg_score**" (read more about **$group** at https://docs.mongodb.com/manual/reference/operator/aggregation/group/). **{$sort: {avg_score: -1}}** operation specifies that the objects generated by the group operation (in the format of {"_id": xx, "avg_score": xx}) should be sorted in descending oder of **avg_score**.

Then in the callback function of the **aggregate** method, use **res.json()** to send to the client side the result prepared by the aggregate method (in the format of [{"_id":xx,"avg_score":xx},{"_id":xx,"avg_score":xx}, ..], which you can check out using console.log(JSON.stringify(result)) before calling **res.json()**).

After this step, the score statistics can be displayed as follows:



Scores Overview

Student UID	Student Name	Exam Name	Score	Update Score
1234	Bob	exam-1	92	Update
1234	Bob	exam-2	89	Update
1234	Bob	exam-3	95	Update
3412	Alice	exam-1	79	Update
3412	Alice	exam-2	88	Update
3412	Alice	exam-3	86	Update
4321	Franck	exam-1	79	Update
4321	Franck	exam-2	99	Update
4321	Franck	exam-3	86	Update

Scores Statistics

Average score of Bob: 92.00.

Average score of Franck: 88.00.

Average score of Alice: 84.33.

Part 3. Implement the update of scores.

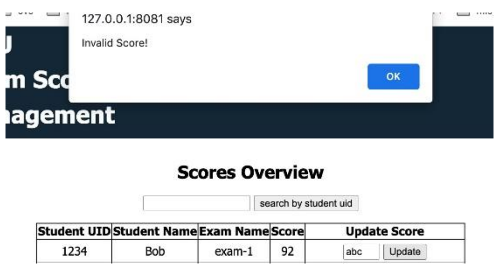
Step 6. Update a score.

6.1. In “**index.js**”, implement the function **updateScore(),** which takes the student uid and the exam name as input. Use **XMLHttpRequest** to open a **POST** request for “**udpate_score**” and set the request header to be ‘**application/json**’. Construct a JSON object with three key-value pairs: (1) “**uid**”: the student uid argument of the function; (2) “**exam**”: the exam name argument of the function; (3) “**new_score**”: the value of the <input> element of id “**new_score_$uid_$exam**”, where **$uid** and **$exam** are the two function arguments, respectively. Finally, use **JSON.stringify()** to convert the JSON object to a JSON string and send it as the request body.

Upon receving the response from the server, first alert the *responseText*, and then refresh the table and re-calculate score statistics by calling **findAllDocs()**.

6.2. In “**app.js**”, note that we have provided the middleware **app.use(express.json())** to parse incoming requests with JSON payloads and populate the *.body* of the request object with the parsed data. Complete the callback function in route **app.post('/update_score', (req, res) => {…})**, that handles the HTTP **POST** request sent by the client when an update button is clicked (the POST request has JSON payload). We first get uid, exam and the new score from the request body (we can use **parseInt()** to convert a string into an integer value). Then check whether the new score value is valid: if the new score is not a number (i.e., **isNaN()** returns true) or it is not within the range [0, 100], we send a response string “Invalid Score!”; otherwise, use collection**.update()** with query **{uid: uid, exam: exam}** and update operation **{$set:{'score':new_score}}** to set “**score”** field of the respective document in the **studList** collection (whose “**uid**” and “**exam**” match the query) to new_score (read more about the update method at https://automattic.github.io/monk/docs/collection/update.html). In the callback function of collection**.update()**, if the error obtained by the update method is *null*, send the string “Successfully updated!” to the client side; otherwise, send the error in the response.

After this step, you can try typing the new score into the input box and clicking “Update”. You are able to see alert messages in your browser in different cases, and the corrsponding “Score” in the table as well as the statistics will be updated accordingly.

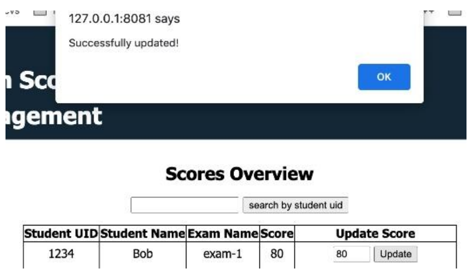


127.0.0.1:8081 says

Invalid Score!

Scores Overview

Student UID	Student Name	Exam Name	Score	Update Score
1234	Bob	exam-1	92	abc Update



127.0.0.1:8081 says

Successfully updated!

Scores Overview

Student UID	Student Name	Exam Name	Score	Update Score
1234	Bob	exam-1	80	80 Update

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 Wednesday March 23, 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.