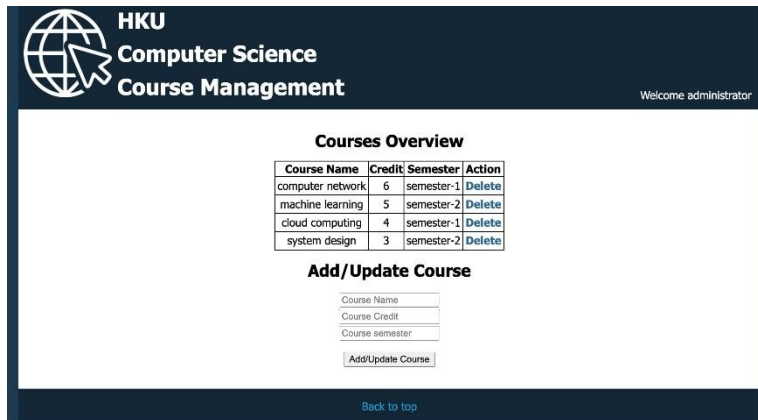


COMP3322B Modern Technologies on World Wide Web

Lab 6: Node.js, Web service, jQuery

Overview

In this lab exercise, we will use Node.js to implement a RESTful Web service, use Pug to generate the HTML page for accessing the Web service, and use jQuery for client-side scripting. In particular, we will use the Express.js Web framework based on Node.js, together with Pug template engine and MongoDB. The Web service allows retrieving, adding, updating and deleting courses in a MongoDB database. The HTML page provides an interface for those operations.



HKU Computer Science Course Management

Welcome administrator

Courses Overview

Course Name	Credit	Semester	Action
computer network	6	semester-1	Delete
machine learning	5	semester-2	Delete
cloud computing	4	semester-1	Delete
system design	3	semester-2	Delete

Add/Update Course

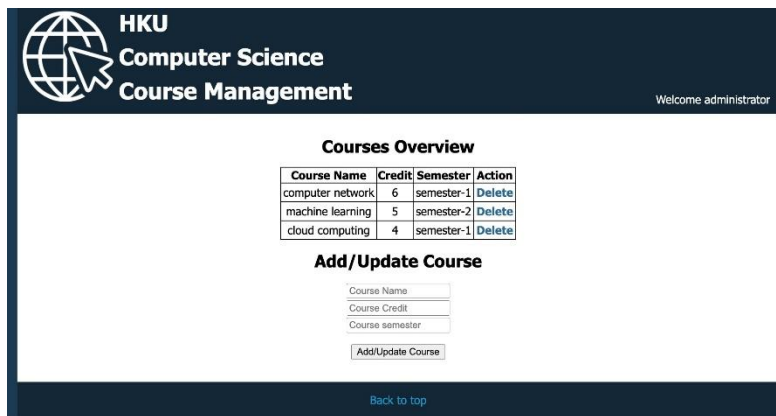
Course Name:

Course Credit:

Course semester:

[Back to top](#)

Upon initial page load, you will see a course table and an add/update module.



HKU Computer Science Course Management

Welcome administrator

Courses Overview

Course Name	Credit	Semester	Action
computer network	6	semester-1	Delete
machine learning	5	semester-2	Delete
cloud computing	4	semester-1	Delete

Add/Update Course


Course Name:

Course Credit:

Course semester:

[Back to top](#)

You can click "Delete" to delete a corresponding course from the database, and it will be removed from the course table. For example, when we delete the course "system design", the table changes accordingly.



HKU
Computer Science
Course Management


Welcome administrator

Courses Overview

Course Name	Credit	Semester	Action
computer network	8	semester-2	Delete
machine learning	5	semester-2	Delete
cloud computing	4	semester-1	Delete

Add/Update Course

Back to top



HKU
Computer Science
Course Management

Welcome administrator

Courses Overview

Course Name	Credit	Semester	Action
computer network	8	semester-2	Delete
machine learning	5	semester-2	Delete
cloud computing	4	semester-1	Delete
convex optimization	6	semester-1	Delete

Add/Update Course

Back to top

You can fill in the three input boxes under “Add/Update Course” and click the “Add/Update Course” button at the bottom to update a course’s information (if you type a course name that exists in the table) or add a new course (if course doesn’t exist) with the input information. For example, after we update the information of “computer network” and add the new course “convex optimization”, the table changes accordingly.

Lab Exercise

Part 1. Preparation

Step 1. Download the code templates from Moodle

Download "**lab6_materials.zip**" from HKU Moodle, and extract it to a folder. In this folder, you will find 5 JavaScript files ("**app.js**", "**index.js**", "**users.js**", "**externalJS.js**", "**generate_db.js**"), 1 CSS file ("**style.css**"), 1 PUG file ("**index.pug**") and 1 Image ("**logo.png**"). In this lab, we will edit "**users.js**", "**externalJS.js**" and "**layout.pug**" (to be renamed from "layout.jade" in the `./views` folder of the Express app), and keep the others of our provided files unchanged. The PUG and JavaScript files can be edited using any code editors or IDEs.

Step 2. Create an Express app

Follow steps 1 to 3 in the handout "setup_nodejs_runtime_and_example_1.pdf" to create an Express app. Use "npm install pug" to install the pug template engine. Rename error.jade, index.jade and layout.jade in `./views` folder to error.pug, index.pug, and layout.pug.

Move files extracted from "**lab6_materials.zip**" to corresponding subdirectories:

- (1) overwrite the original "**app.js**" in the Express app directory with the "**app.js**" we provided.
- (2) overwrite the original "**index.js**", "**users.js**" in `./routes` with the same files that we provided;
- (3) move "**externalJS.js**" to `./public/javascripts`;
- (4) move "**style.css**" to `./public/stylesheets`;
- (5) overwrite the original "**index.pug**" with the same file that we provided;
- (6) move "**logo.png**" to `./public/images`.

"**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 (keep the MongoDB server running in the terminal).

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 "lab6-db" in the database server

and inserts a few course documents into a **courseList** collection in the database. Each document in the **courseList** collection will contain 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 inserted documents using **db.courseList.find()** in the interactive shell (refer to step 4 of Example 6 in the handout “AJAX_JSON_MongoDB_setup_and_examples.pdf”).
- **name**: The name of the course
- **semester**: The offering semester of the course
- **credit**: The credit of the course

All the values are strings.

Now go to the express app directory, and use “npm install --save monk” to install the monk module for your project.

Step 4. Open **app.js** in an editor and check out its content. You should be able to understand the code according to the lecture slides and what we introduce in “setup_nodejs_runtime_and_examples_2.pdf”. You can see a number of middlewares have been included to handle the incoming requests, among which we will need `express.json()`, `express.urlencoded()` and `express.static()` in this lab. Besides, the “morgan” module is used for logging the request status for development usage, which you will see on the terminal where you run the server app.

This line of code “`module.exports = app;`” at the end of **app.js** exports this app as the default module that the Express app will run once started; then you can start the Express app by typing “`npm start`” in the your express app directory.

After running “`npm start`”, the web server is started and listens at the default port 3000. You can launch a web browser and visit the web page at <http://127.0.0.1:3000> or <http://localhost:3000>.

Part 2: Create the Web Page Using Pug

We next modify the Pug template in the `./views` directory of your express app, in order to render the homepage of the app.

Step 5. Open **index.pug** using an editor. Please refer to <https://pugjs.org/api/getting-started.html> to understand the code in the file.

Step 6. Open **layout.pug** using an editor and modify it to contain the following content:

```

doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
    meta(name="viewport" content="width=device-width, initial-scale=1.0")
  body
    block content
    script(src='https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js')
    script(src='/javascripts/externalJS.js')

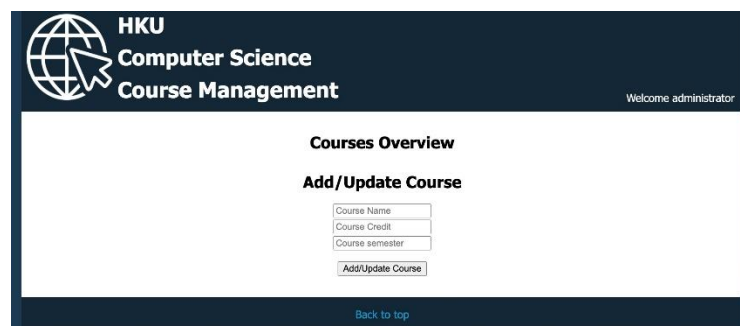
```

The first line of code in **index.pug** indicates that **index.pug** extends **layout.pug**. By modifying **layout.pug** as above, the web page rendered links to:

- (1) a **style.css** file under [./public/stylesheets](#) for styling;
- (2) the jQuery library on Google server;
- (3) an **externalJS.js** file under [./public/javascripts](#) containing client-side JavaScript.

Note that the [./public](#) directory has been declared to hold static files which can be directly retrieved by a client browser, using the line of code “`app.use(express.static(path.join(__dirname, 'public')));`” in **app.js** (note that there are two underscores “`_`” before `dirname` in the code). In this way, the rendered web page can directly load files under the [./public](#) directory.

Now let’s check out the web page rendered using the Pug files. Use “`npm start`” to start the Express app (**you should always use control+C to kill an already running app before you start the app again after making modifications**). Check out the rendered page at <http://localhost:3000> on your browser. You should see a page like the following.



Especially, your GET request for “/” is handled on the server side by the router **index.js**, which renders the HTML page using **index.pug** and **layout.pug**. Further, on the terminal where you have run “`npm start`”, you can see prompts like the following (logged by the “morgan” module), showing that the browser has issued three more GET requests (after the GET request for “/”) to the server side to retrieve the client-side javascript file, the logo image file and the styling sheet file, respectively, which were linked to in **layout.pug** and **index.pug**.

```
GET / 304 605.500 ms - -
GET /javascripts/externalJS.js 304 1.365 ms - -
GET /images/logo.png 304 1.186 ms - -
GET /stylesheets/style.css 304 0.300 ms - -
```

Part 2. Implement courses overview and deletion

Step 7. Show courses information in a table

In this step, we implement server-side and client-side code for retrieving course information from the database and display them in a table on the web page shown on the client browser, when the page is loaded.

7.1. In the server-side script `users.js`, complete the callback function in `router.get('/get_courses', (req, res) => {...})`, which specifies how the server responds to the HTTP GET requests for http://localhost:3000/users/get_courses. The middleware uses `collection.find()` to find all documents in the `courseList` collection, and use `res.json()` to send a JSON array containing all the course documents found.

You can restart your Express app with “npm start” in the terminal. Test if your above server-side code works by browsing http://localhost:3000/users/get_courses on your browser. The browser should display a JSON response text like this:

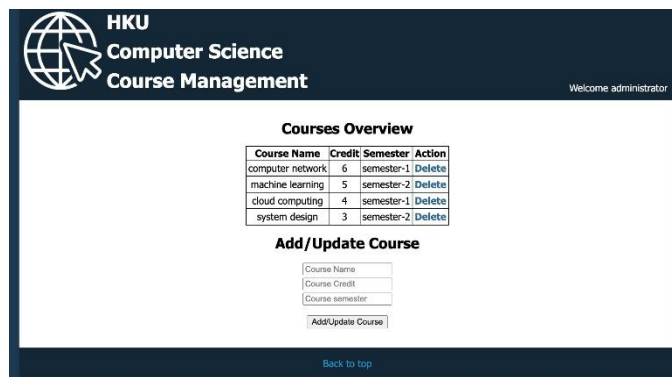
```
[{"_id":"624158c259150c9f2ae2184c","name":"computer network","semester":"semester-1","credit":"6"}, {"_id":"624158c259150c9f2ae2184d","name":"machine learning","semester":"semester-2","credit":"5"}, {"_id":"624158c259150c9f2ae2184e","name":"cloud computing","semester":"semester-1","credit":"4"}, {"_id":"624158c259150c9f2ae2184f","name":"system design","semester":"semester-2","credit":"3"}]
```

7.2. Now we add client-side code for displaying the course table. Recall that in Step 6, the rendered HTML page is linked to `externalJS.js`. Open `externalJS.js` in an editor and check out the code provided. We are going to implement functions in `externalJS.js` using jQuery. The jQuery code is executed when the page has been loaded by the browser (`$(document).ready()`). Especially, we call the function `showAllCourses()` when the page is loaded, to add class information retrieved from the server side into a course table.

Complete the function `showAllCourse()`. The table header has been stored in the string `table_content`. Use `$.getJSON()` method with url `"/users/get_courses"` to send a HTTP GET request to the server side. As implemented in Step 7.1, the server will return a JSON array, in which each object is a course document. In the callback function of `$.getJSON()`, store the

received JSON array in a global variable *courses_list* (it will be used in later steps), and then iterate over the array using `$.each()`. For each document in the array, create the HTML representation of a table row with four `<td>` elements: the values of the first three `<td>` elements are “**name**”, “**credit**”, “**semester**” fields of the course document, which can be obtained by **this.name**, **this.credit** and **this.semester**, respectively; the last `<td>` element contains an `<a>` element with text “Delete” and attributes `href="#" class="linkDelete" rel="{this._id}"` (i.e., we are creating the Delete link as shown in the screenshots at the beginning of this handout). The table row representations should be all concatenated into the string *table_content*. Finally, use `$('#course_table').html()` to set content of the table element of id “course_table” to *table_content*.

Now browse the web page again at <http://localhost:3000/>. You should be able to see all courses shown in a table like the following (note that the “Delete” link is not bound to event handler yet):



Step 8. Delete a course by clicking “Delete” in the same row.

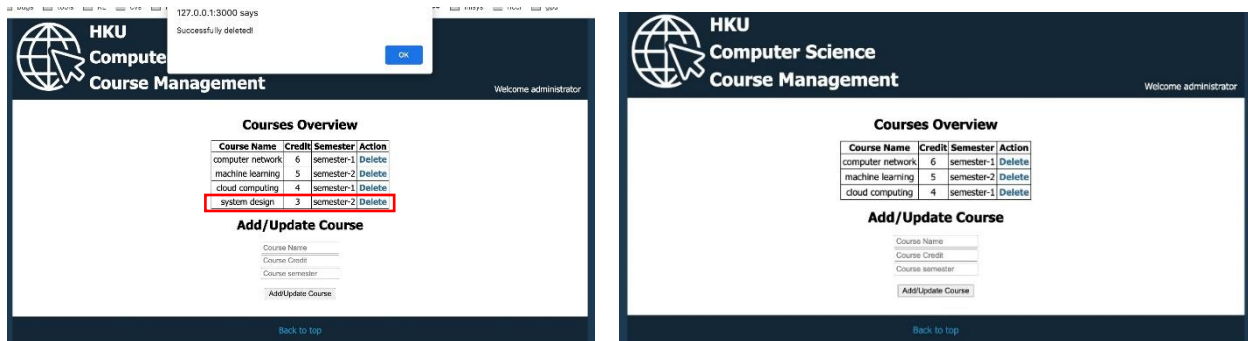
We next implement server-side and client-side code for deleting a course from the database, as well as from the table on the web page shown on the client browser, when a “Delete” link on a table row is clicked.

8.1. In “*users.js*”, complete the callback function in `router.delete('/delete_course/:id', (req, res) => {...})`, which specifies how the server responds to the HTTP DELETE requests for http://localhost:3000/users/delete_course/somecourseid. The middleware retrieves `_id` of the course that the client wants to delete from `req.params.id`, and uses `collection.remove({'_id': req.params.id})` to remove the course document from the **courseList** collection. In the callback function of `collection.remove()`, if the error of the **remove** operation is *null*, send the string “Successfully deleted!” to the client side; otherwise, send the error in the response.

8.2. In “*externalJS.js*”, you can find that we have registered a handler function `deleteCourse` to the click event on each “Delete” link in the table of id “course_table” by selecting elements of class “linkDelete” in the table: `$("#course_table").on('click', '.linkDelete', deleteCourse)`. Please refer to <https://api.jquery.com/on/> to understand more of how we use the second parameter of `.on()` to filter the descendants of the selected elements that trigger the event. Complete the event

handler function **deleteCourse(event)**: we have used **event.preventDefault()** to prevent opening the link “#” when the hyperlink is clicked; retrieve **_id** of the course that you are going to delete from the ‘rel’ attribute using **\$(this).attr('rel')**); then use **\$.ajax()** to send a HTTP **DELETE** request for **“/users/delete_course/:id”** (*id* is the course **_id**); upon receiving the server response, alert the response message and call **showAllCourses()** to refresh the course table.

Restart your Express app and browse <http://localhost:3000>. You are able to delete a course by clicking the “Delete” link in the same row. The page will show up like the following when you delete the course “system design”:



Part 3. Implement Update/Add of a course

Step 9. Update an existing course with input information.

We next implement server-side and client-side code for updating an existing course in the database, and have the updated course information shown in the course table in the web page on the client browser.

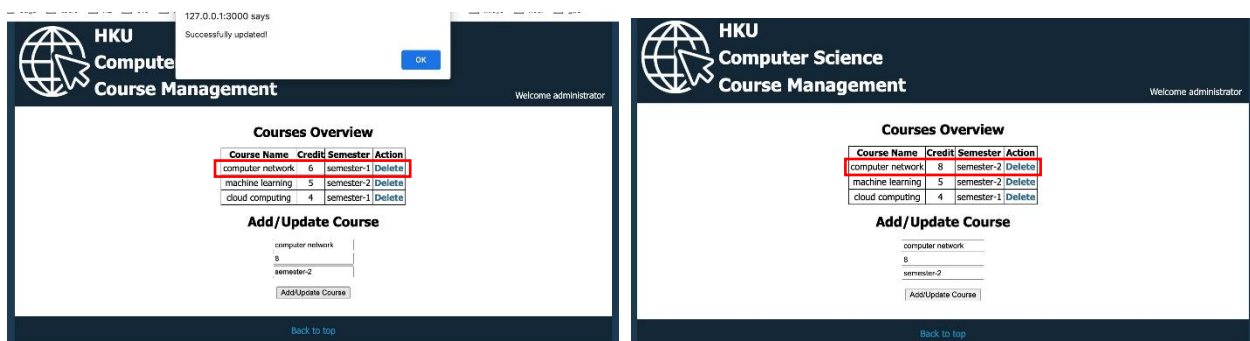
9.1. In “users.js”, complete the callback function of **router.put('/update_course/:id', (req, res) => {...})**, which specifies how the server responds to the HTTP PUT requests for http://localhost:3000/users/update_course/somecourseid. The middleware uses **collection.update()** with query **{“_id”: req.params.id}** and update operation **{\$set:{'name':req.body.name, 'credit': req.body.credit, 'semester': req.body.semester}}** to set the fields of the respective course document in the **courseList** collection (whose “_id” matches the query). In the callback function of **collection.update()**, if the error of the update operation is **null**, send the JSON object **{msg: “Successfully updated!”}** to the client side; otherwise, send **{msg: error}** in the response.

9.2. In “externalJS.js”, you can find that we have registered an event handler **addOrUpdateCourse** to the click event of the “Add/Update Course” button. In the event handler function **addOrUpdateCourse()**, we have called **event.preventDefault()** to prevent any default action, and used the provided function **isInputFilled()** to check whether all input textboxes are filled with contents. If any textboxes are not filled, we alert “Please fill in all fields.” If all three

textboxes are filled, we use the input contents to form a new JSON object *new_doc* with three key-value pairs: `{"name": $("#inputName").val(), "credit": $("#inputCredit").val(), "semester": $("#inputSemester").val() }`. We further apply a map function to the global array `courses_list` (which stores information of existing courses locally), obtain a new array containing all existing courses' names (learn more about the map at https://www.w3schools.com/jsref/jsref_map.asp), and then identify the index of the input name in the array.

Complete the code under the condition *index!=-1* (which means the course exists and we are updating the course information). Use `$.ajax()` to send a HTTP **PUT** request for `"/users/update_course/courseid"`, where the course id is the `"_id"` field of the course document (corresponding to the input course name). The request should use JSON as `dataType` and the new JSON object *new_doc* as data in the body. Upon receiving response from the server (which is a JSON object containing a `"msg"` key), alert the value of `"msg"` and call `showAllCourses()` to refresh the table.

Restart your Express app and browse <http://localhost:3000>. You are able to update an existing course by filling the input textboxes and clicking the "Add/Update Course" button. The page will show up like the following when updating information of "computer network":



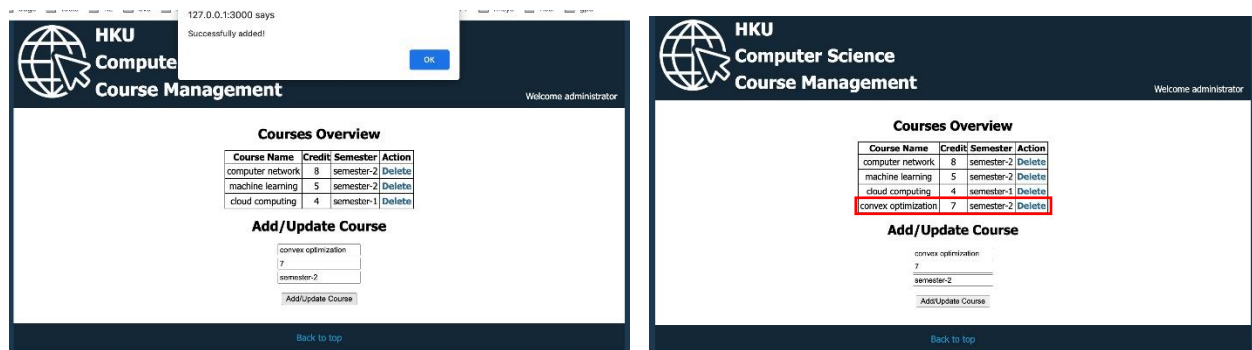
Step 10. Add a new course with input information.

We now implement server-side and client-side code for adding a new course into the database, as well as showing the new course in the course table on the web page on the client browser.

10.1. In `"users.js"`, complete the callback function of `router.post('/add_course', (req, res) => {...})`, which specifies how the server responds to the HTTP POST requests for http://localhost:3000/users/add_course. The middleware uses `collection.insert()` to add `req.body` as a new document in the `courseList` collection (read more about `insert` at <https://automattic.github.io/monk/docs/collection/insert.html>). In the callback function of `collection.insert()`, if the error of the `insert` operation is `null`, send the JSON object `{msg: "Successfully added!"}` to the client side; otherwise, send `{msg: error}` in the response.

10.2. In “externalJS.js”, complete the code in the *else* branch of *if(index != -1)* (i.e., the case of *index==-1*, which means that the course does not exist and we will add a new course using the input information). Use **\$.ajax()** to send a HTTP **POST** request for “/users/add_course”. The request uses JSON as the *dataType* and the new JSON object *new_doc* as data in the body. Upon receiving response from the server (which is a JSON object containing a “msg” key), alert the value of “msg” and call **showAllCourses()** to refresh the table.

Restart your Express app and browse <http://localhost:3000> again. You are able to add a new course by filling the input textboxes and clicking the “Add/Update Course” button. The page will show up like the following when adding a new course “convex optimization”:



Congratulations! Now you have finished Lab 6. 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 April 6, 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 6”.
- (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.