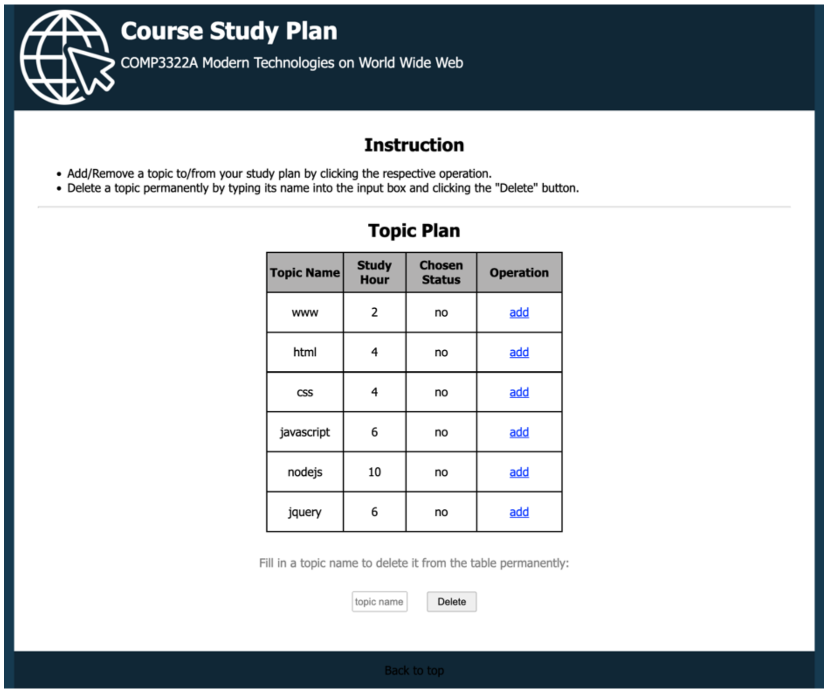
COMP3322A Modern Technologies on World Wide Web
Lab 6: Node.js, Web service, Pug

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, updating and deleting topics in a MongoDB database. The HTML page provides an interface for those operations.



Course Study Plan

COMP3322A Modern Technologies on World Wide Web

Instruction

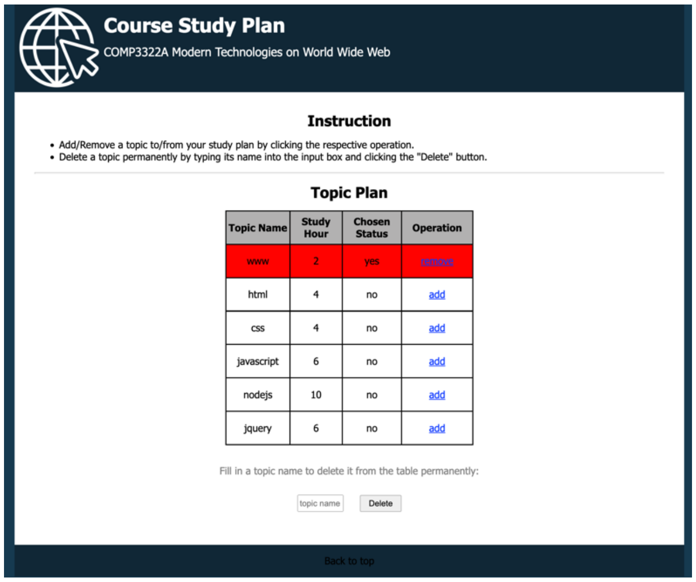
- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

Topic Name	Study Hour	Chosen Status	Operation
www	2	no	add
html	4	no	add
css	4	no	add
javascript	6	no	add
nodejs	10	no	add
jquery	6	no	add

Fill in a topic name to delete it from the table permanently:

Upon initial page load, you will see a topic table. Each topic has a recommended study hour per week and a status ("yes" or "no") indicating whether you have chosen it into your study plan.



Course Study Plan

COMP3322A Modern Technologies on World Wide Web

Instruction

- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

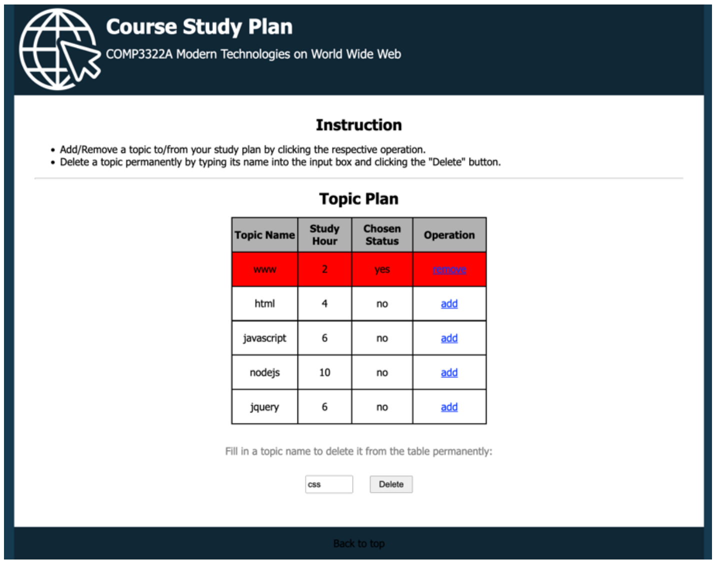
Topic Name	Study Hour	Chosen Status	Operation
www	2	yes	remove
html	4	no	add
css	4	no	add
javascript	6	no	add
nodejs	10	no	add
jquery	6	no	add

Fill in a topic name to delete it from the table permanently:

topic name Delete

Back to top

You can click "add"/"remove" operations to update your study plan. For example, when you click the "add" link of the topic "www", the row of the topic will be highlighted in red, status will change to "yes", and the operation will change from "add" to "remove", which is also clickable to remove the topic from your study plan.



Course Study Plan

COMP3322A Modern Technologies on World Wide Web

Instruction

- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

Topic Name	Study Hour	Chosen Status	Operation
www	2	yes	remove
html	4	no	add
javascript	6	no	add
nodejs	10	no	add
jquery	6	no	add

Fill in a topic name to delete it from the table permanently:

css Delete

Back to top

Below the table, you can fill in a valid topic name into the input box and delete it permanently from the table (i.e., delete from the server-side database) by clicking the "Delete"button.

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 "**lab6_materials.zip**" from HKU Moodle, and extract it to a folder. In this folder, you will find 4 JavaScript files (“**app.js**”, “**users.js**”, “**externalJS.js**”, “**generate_db.js**”), 1 CSS file (“**style.css**”), 2 PUG files (“**index.pug**” and “**layout.pug**”) and 1 Image (“**logo.png**”). In this lab, we will only edit “**users.js**” and “**externalJS.js**”, and keep the others of our provided files unchanged.

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 “**users.js**” in ./routes with the same file that we provided;
(3) move “**externalJS.js**” to ./public/javascripts;
(4) move “**style.css**” to ./public/stylesheets;
(5) overwrite the original “**index.pug**” and “**layout.pug**” with the same files 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 (only 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 (then 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 topic documents into a **topicList** collection in the database. Each document in the **topicList** collection will contain the following key-value pairs:

- **_id**: The unique ID of the document, which the MongoDB server adds automatically into each inserted document. You can check **_id** of inserted documents using **db.topicList.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 topic.
- **hour**: The study hour required of the topic (integer format).
- **status**: a "yes" or "no" string indicating whether the topic is added into your study plan.

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 content and what we explain in the handout "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 templates 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 check out its content, as follows:

```
doctype html
html
  head
    title= "lab6"
    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/3.6.0/jquery.min.js')
    script(src='/javascripts/externalJS.js')
```

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

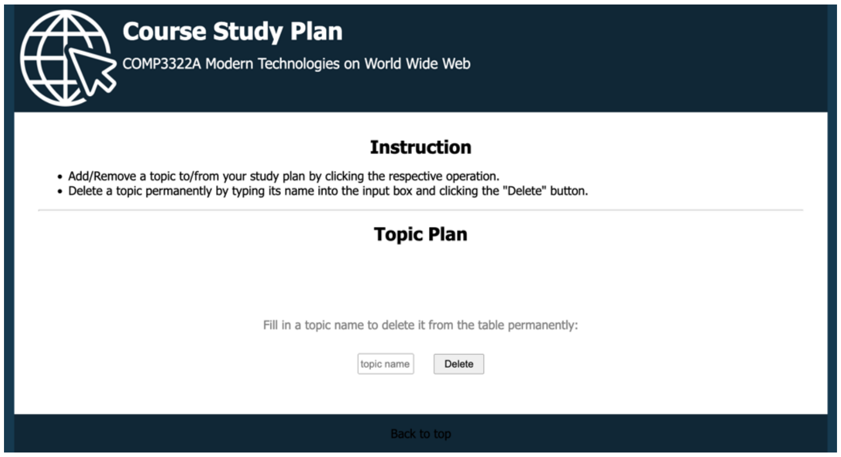
(1) a **style.css** file under ./public/stylesheets for styling;

(2) the jQuery library on Google CDN server;

(3) an **externalJS.js** file under ./public/javascripts containing client-side JavaScript code.

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.



Course Study Plan

COMP3322A Modern Technologies on World Wide Web

Instruction

- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

Fill in a topic name to delete it from the table permanently:

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 3. Implement topic overview and operations

Step 7. Show topics information in a table

In this step, we implement server-side and client-side code for retrieving topic 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_table', (req, res) => {…})**, which specifies how the server responds to the HTTP GET requests for http://localhost:3000/users/get_table. The middleware should use **collection.find()** to find all documents in the **topicList** collection, and use **res.json()** to send a JSON array containing all the topic documents found to the client.

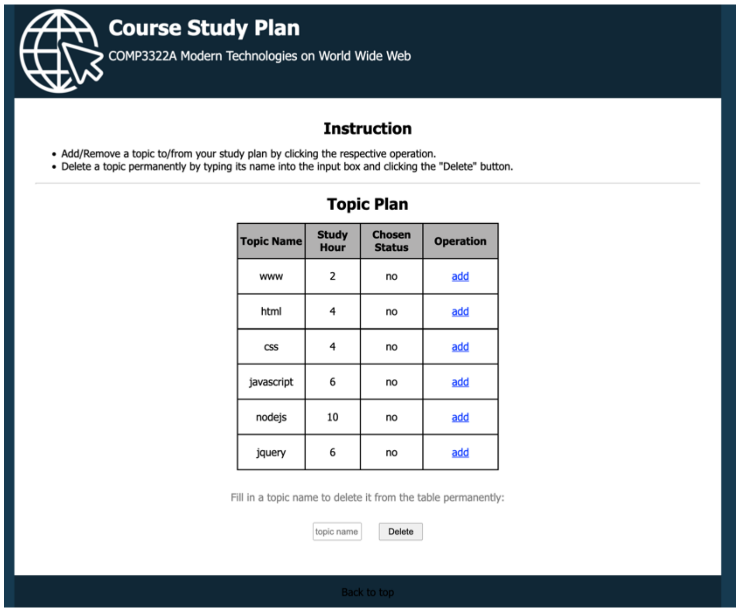
You can restart your Express app with "npm start" in the terminal. Test if your above server-side code works by entering http://localhost:3000/users/get_table in the address bar of your browser. The browser should display a JSON response text like this:

```
[{"_id":"636898c943c18d902a1cfcef","name":"www","hour":2,"status":"no"},{"_id":"636898c9
43c18d902a1cfcf0","name":"html","hour":4,"status":"no"},{"_id":"636898c943c18d902a1cfcf1"
,"name":"css","hour":4,"status":"no"},{"_id":"636898c943c18d902a1cfcf2","name":"javascript",
"hour":6,"status":"no"},{"_id":"636898c943c18d902a1cfcf3","name":"nodejs","hour":10,"statu
s":"no"},{"_id":"636898c943c18d902a1cfcf4","name":"jquery","hour":6,"status":"no"}]
```

7.2. Now we add client-side code for displaying the topic table. Recall that in Step 6, the rendered HTML page is linked to **externalJS.js**. Open **externalJS.js** 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 **showAllTopics()** when the page is loaded, to add topic information retrieved from the server side into a topic table.

Complete the function **showAllTopics()**. The table header has been stored in a string *table_content.* Use **$.getJSON()** method with url "**/users/get_table**" to send a HTTP **GET** request to the server side. As implemented in Step 7.1, the server will return a JSON array, in which each object is a topic document. In the callback function of **$.getJSON()**, iterate over this array using **$.each()**. For each document in the array, create the HTML representation of a table row with four <td> elements: the values of the first three <td> elements are "**name**", "**hour**", "**status**" fields of the topic document; the last <td> element contains an <a> element with text “**add**” or “**remove**”(if status field equals to “**yes**”, text should be “**remove**”; otherwise, text should be “**add**”), and attributes *href="#" class="operation" rel=_id field of the topic document* (i.e., we are creating the add/remove link as shown in the screenshots at the beginning of this handout). Also for each row tag, add the attribute class=“highlight” if the status field of the topic is “**yes**”. The table row representations should be all concatenated into the string *table_content*. Finally, use **$(‘#plan_table’).html()** to set HTML content of the table element of id “plan_table” to *table_content*.

Now browse the web page again at http://localhost:3000/. You should be able to see all topics shown in a table like the following (note that the “add”/“remove” link is not bound to event handler yet, and no topic is selected into your study plan yet):



Course Study Plan

COMP3322A Modern Technologies on World Wide Web

Instruction

- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

Topic Name	Study Hour	Chosen Status	Operation
www	2	no	add
html	4	no	add
css	4	no	add
javascript	6	no	add
nodejs	10	no	add
jquery	6	no	add

Fill in a topic name to delete it from the table permanently:

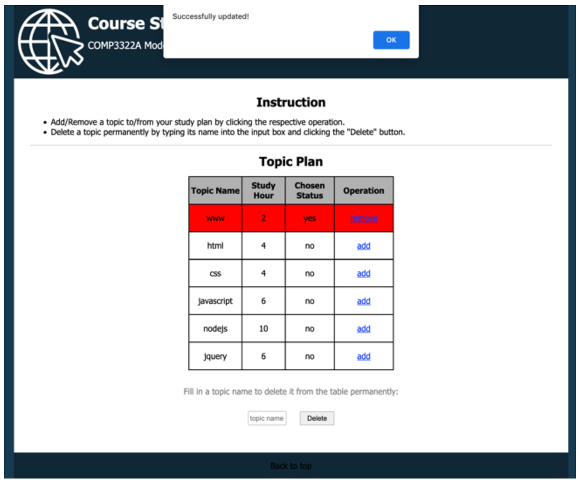
Step 8. Add/remove a topic to/from your study plan by clicking the operation link in the same row.

We next implement server-side and client-side code for adding/removing a topic to/from your study plan, when the “add” or “remove” link on a table row is clicked.

8.1. In “**users.js**”, complete the callback function in **router.put(‘/update_status/’, (req. res) => {…}),** which specifies how the server responds to the HTTP PUT requests for http://localhost:3000/users/update_status. Use **collection.update({‘_id’: req.body._id}, {$set:{status: new_status}})** to update the status field of the topic with id **req.body._id**. The **new_status** should be “**yes**” if the operation (retrieved from **req.body.op**) is “**add**”; otherwise, if the operation is “**remove**”, new_status should be “**no**”. If the **update** operation is successful, send “Successfully updated!” to the client; otherwise, send the error in the response.

8.2. In “**externalJS.js**”, you can find that we register a handler function **operateTopic()** to the click event on each operation link in the table of id “**plan_table**” by: **$("#plan_table").on('click', '.operation', operate).** Please refer to https://api.jquery.com/on/ to understand more of how we use the second parameter of **.on()** to filter the descendants of the selected elements that trigger the event. Complete the event handler function **operateTopic(event)**: we have used **event.preventDefault()** to prevent opening the link “#” when the hyperlink is clicked; retrieve **_id** of the topic that you are going to add/remove from the ‘rel’ attribute using **$(this).attr(‘rel’)** and retrieve the operation that your are going to perform using **$(this).html()**. Then use **$.ajax()** to send a HTTP **PUT** request for “**/users/update_status**” with JSON data **{_id: _id field retrieved, op: operation retrieved};** upon receiving the server response, alert the response message and call **showAllTopics()** to refresh the topic table.

Restart your Express app and browse http://localhost:3000. You are able to add/remove a topic to/from your study plan by clicking the “add”/“remove” link in the same row. Only when a topic is in your study plan (i.e., status field equals to “yes”), its row is highlighted in red.



Successfully updated!

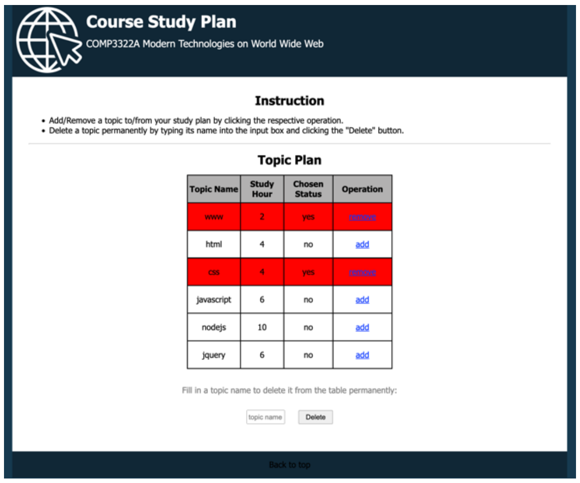
Instruction

- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

Topic Name	Study Hour	Chosen Status	Operation
www	2	yes	remove
html	4	no	add
css	4	no	add
javascript	6	no	add
nodejs	10	no	add
jquery	6	no	add

Fill in a topic name to delete it from the table permanently:



Course Study Plan

COMP3322A Modern Technologies on World Wide Web

Instruction

- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

Topic Name	Study Hour	Chosen Status	Operation
www	2	yes	remove
html	4	no	add
css	4	yes	remove
javascript	6	no	add
nodejs	10	no	add
jquery	6	no	add

Fill in a topic name to delete it from the table permanently:

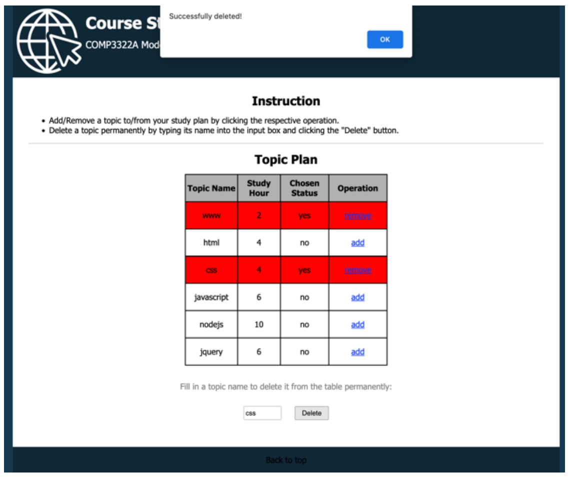
Step 9. Delete a topic from the table permanently.

9.1. In “users.js”, complete the callback function in **router.delete(‘/delete_topic/:name’, (req. res) => {…}),** which specifies how the server responds to the HTTP DELETE requests for http://localhost:3000/users/delete_topic/xx (where xx is the name of the topic). The middleware retrieves name of the topic that the client wants to delete from *req.params.name*, and uses collection**.remove({‘name’: req.params.name})** to remove the topic document from the **topicList** collection. If the **remove** operation is successful, send the string “Successfully deleted!” to the client side; otherwise, send the error in the response.

9.2. In “**externalJS.js**”, you can find that we have registered a handler function **deleteTopic()** to the click event on the “Delete” button underneath the topic table, by selecting element with id

“#submit_delete”. Complete the event handler function **deleteTopic(event)**: retrieve the topic name to be deleted by obtaining the value of input box with **$(“#input_name”).val()**; check whether the topic name retrieved is an existing name in the topic table by using **$(`td:contains(“${topic_name}”)`),** which gets a list of <td> elements whose innerhtml contains the topic name (ref to this link for contains() selector: https://www.w3schools.com/jquery/sel_contains.asp). If the length of the list is not 0, use **$.ajax()** to send a HTTP **DELETE** request for “**/users/delete_topic/:topic_name**”; otherwise, alert the message “No such topic in the table!”. Upon receiving the server response, alert the response message and call **showAllTopics()** to refresh the topic table.

Restart your Express app and browse http://localhost:3000. You are able to delete a topic by typing its name and clicking the “Delete” button. The page will show up like the following:



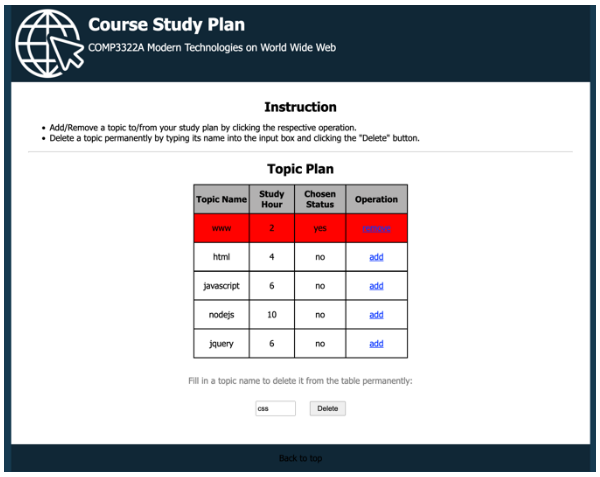
Instruction

- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

Topic Name	Study Hour	Chosen Status	Operation
www	2	yes	remove
html	4	no	add
css	4	yes	remove
javascript	6	no	add
nodejs	10	no	add
jquery	6	no	add

Fill in a topic name to delete it from the table permanently:



Instruction

- Add/Remove a topic to/from your study plan by clicking the respective operation.
- Delete a topic permanently by typing its name into the input box and clicking the "Delete" button.

Topic Plan

Topic Name	Study Hour	Chosen Status	Operation
www	2	yes	remove
html	4	no	add
javascript	6	no	add
nodejs	10	no	add
jquery	6	no	add

Fill in a topic name to delete it from the table permanently:

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 Thursday November 17, 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.