Task

The goal of this coursework is to build the back end for the app created in the first coursework. MongoDB will be used for storing the data, and data exchange between the app and the database will be done through REST API implemented using Express.js. The back end will run on a Node.js server.

Requirements

A submission will receive zero marks if it fails any of the following requirements:

- The backend server must use ‘Node.js’; Apache or Xampp is not allowed;
- All database access, such as storing and retrieving data, must be achieved through ‘REST API’; any other type of access is not allowed, including direct database access;
- The REST API must be developed with ‘Express.js’;
- The front-end data access must be achieved with ‘promise’ using ‘fetch’ function; ‘XMLHttpRequest’ or library such as axios:js is not allowed.
- The code must be hosted in a GitHub repository with at least 10 commits. This must be a new repository; you cannot use the repository for CW1.
- The web app must be demonstrated. Coursework code will not receive any mark, even it works fine if it cannot be explained satisfactorily during the in-lab demonstration, i.e., a student cannot explain what the code does.

Create a server using Express.js
Maximum score
2

Create two 'get' routes: when the path is '/lessons', the server returns a list of lessons as shown in the module handbook
Maximum score
2

When the path is '/user', the server returns the information of an user as shown in the module handbook
Maximum score
2

Create a Vue app that retrieves the lesson list from the Express server with Fetch. 'XMLHttpRequest' and library such as 'axios.js' are not allowed
Maximum score
2

The client code then displays the list of lesson using Vue similar to the screenshot in the module handbook
Maximum score
2

A submission will receive zero marks if it fails any of the following requirements:

- The data must be stored in 'MongoDB Atlas'; local MongoDB or any other database is not allowed;
- Connection to MongoDB must use the native Node.js driver only; libraries like Mongoose is not allowed.
- The Node/Express server must be hosted on Heroku.com; a local server is not allowed.

Submission

Include the following files in one zip file that is no more than 10MB.

- A text file include the URLs to:
  - The GitHub repository of your back end code (Node and Express)
  - The GitHub Page that runs your front end (the Vue.js code)
  - The Heroku route that returns the information of lesson information.
- The 'lesson' and the 'order' collection in the MongoDB Atlas. See here for how to export collection in MongoDB Compass (https://docs.mongodb.com/compass/current/import-export#export-data-from-a-collection).
- The back end code (Node and Express): do not include the 'node modules' folder. Otherwise, the zip will be too big to submit.
- The front end code (updated Vue.js from CW1 with data now fetched from MongoDB)
- The requests created in Postman. See here for how to export requests in Postman (https://learning.postman.com/docs/getting-started/importing-and-exporting-data/#exporting-collections).

MongoDB
– Have a MongoDB collection for lesson information (2%) - minimal fields: topic, price, location, and space
– Have a MongoDB collection for order information (2%) - minimal fields: name, phone number, lesson ID, and number of space.
Maximum score
4

Middleware
- Create a 'logger' middleware that output all requests to the server console (2%)
- Create a static file middleware that returns lesson images or an error message if the image file does not exist. (2%)
Maximum score
4

REST API
– A GET route that returns all the lessons (2%)
– A POST route that saves a new order to the 'order' collection (2%);
– A PUT route that updates the number of available spaces in the 'lesson' collection after an order is submitted (2%).
– At least one Postman request is created for each route (2%).

Maximum score
8

Fetch
– A fetch that retrieves all the lessons with GET.
– A fetch that saves a new order with POST after it is submitted.
– A fetch that updates the available lesson space with PUT after an order is submitted.
Maximum score
4

Search
– This is the challenge component of this coursework, and it is not expected that everyone can complete it. The solution is not covered in the lecture or lab, so you need to research it.
– The goal is to add a full-text search feature, similar to the challenge component of the Coursework 1. The difference is that the search needs to be performed in the back end (Express + MongoDB), not in the front end as in Coursework 1 (Vue + JavaScript). You will not receive any mark for this part if the search is performed in the front end.
– You cannot use any existing library to implement this function. Otherwise, you will not receive any mark for this part.
– Fetch (2%): in the front end, a ‘fetch’ request should be created to created to send the search information to the back end.
– Express API (2%): a Express.js route should be created to handle the search request and return the search results from the MondoDB.
– Search as you type (1%): similar to Coursework 1, there is one mark if the search supports ‘search as you type’, i.e., the search starts when user types the first letter (displaying all the lessons containing that letter) and the result list is filtered as more search letters are entered (similar to Google search).
Maximum score
5