DA219A - Lab 1

LABORATION'S GOAL

Learn how to integrate MongoDB with Node.js and create a RESTful API using Express to retrieve data from the MongoDB database.

PRE-REQUIREMENTS

In this lab, you will need the following:

- Node.js
- Nodemon
- Express
- A MongoDB database
- Postman or any similar tool

For this lab, you should install these components separately in your project (although you should install "Nodemon" globally).

PREPARATION OF THE DEVELOPMENT ENVIRONMENT:

(Skip this if you have Node.js Express.js and Nodemon installed BUT make sure to add the required scripts)

Npm

If you have not installed npm and node earlier. Please refer to the following videos for guidance.
Select one of the following videos based upon your operating system.

Linux
Node.js and npm download
https://www.youtube.com/watch?v=P6rPI7x28nY

Windows
Node.js and npm download
https://www.youtube.com/watch?v=X-FPCwZFU_8

Mac
Node.js and npm download
https://www.youtube.com/watch?v=0i-gstqgjuE

NODE. JS

After installing Node.js on your computer, create a simple project that you can extend it later. Create Http server and show some text on the browser. Use the port 3000. Name the file index.js. Run your app using node index.js

🛠️ Create a directory for your project, cd into the directory, and initialize a Node project with default settings:

$ npm init

This will create a package.json file in the directory. Add the git repository at this step or later if you prefer.

Express

Install express and create a new file **index.js** for the server!

NODEMON

🛠️Install Nodeman globally

```
npm i -g -D nodemon
```

After installing Nodemon, start your project with “nodemon” instead of “node” and verify that any change to a Node.js source file (In this case app.js) will also be reflected in the browser without restarting the server.

Now instead of starting the mini webserver app.js using node index.js use:
nodemon index.js

You can setup a new npm script to make your workflow much more easier.
Edit your package.json

```
"scripts": {
        "start": "node index.js",
        "dev": "nodemon index.js"
}
```

So, if you used npm start , it will run your app. If you used npm dev, nodemon will run and any changes will be reflected automatically.

MongoDB Preparation

🛠️ Sign up for a free tier database from a cloud service provider. MongoDB provides a free tier of their cloud database (MongoDB Atlas).

To create a database on MongoDB Atlas, you will need:

- Make sure your IP is on the Whitelist.
- Make sure you have a user account and password on the MongoDB cluster you want to use.
- Open a browser and log in to https://cloud.mongodb.com

Once you have access to a database, create a new collection and add a few documents of data to it.

There are some DBMS GUI:s you may download freely, such as MongoDB Compass or TablePlus.
To install compass use the link install compass

Integrate your database to node.js

Now add the corresponding database driver to your Node.js project and verify that you can connect to the database and show some content from your database in the browser. It is recommended to use Mongoose:

🛠️ npm install mongoose

This guide will help you integrate your database to Express:
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

TASKS

1. Create git repository for the lab to publish your code.
2. Create a database and add a new collection of music albums. The Album should at least have an id, title, artist name, and year. Add some albums to your collection.
3. Create an .env file and add the PORT and the CONNECTION URL to it.(The connection string should be there, hard coding it will not be accepted)
4. Create. gitignore file and add node_modules,. env, .vscode, .git to ignore pushing them to github
5. Add a file to your project called "index.html". When you start your Node.js application, you shall be able to see it in your browser, i.e., going to http://localhost:3000/ shall bring you to the "homepage" of your Node.js server. In this page, you will show the data from the database you created before. Read this for guidance https://www.digitalocean.com/community/tutorials/use-expressjs-to-deliver-html-files
6. Add CRUD operations to your REST API for the music albums collection. You should be able to:

    a. **Show all albums** (GET http://server:port/api/albums) should return a JSON array with all the albums.

    b. **Retrieve a specific album by title** (GET http://server:port/api/albums/:title) should return a JSON object with the attributes of the album
    (for example, a JSON object for the album with the title= "Abbey Road" should be returned if we used the URL http://localhost:3000/api/albums/Abbey Road .
    It can look like the following:

    ```
    {
     "id": 1,
    "title": "Abbey Road",
     "artist": "The Beatles",
    "year": 1969
    }
    ```

        i. If the {title} is not found, an error message should be returned as well as HTTP-status code 404.
        ii. If more than one albums have the same title, all should be returned

    c. **Create a new album** (POST http://server:port/api/albums)
        i. If the album is not added before (not found in the database), the new album should be added to the database and a JSON

object of the newly added album should be returned as well as HTTP-status code 201 (created).

ii. If the album already exists in the database, an error message should be returned as well as HTTP-status code 409 (conflict).

d. Update an album (PUT http://server:port/api/albums/:id)

i. If the {id} is not found, an error message should be returned as well as HTTP-status code 404.

e. Delete an album (DELETE http://server:port/api/albums/:id)

i. **If the {id} is not found, an error message should be returned as well as HTTP-status code 404.**

To read about different CRUD operations
https://www.apinewbies.com/api-request/

7. Populate your index.html file with the API data
(Remember to use JSON.stringify(data).)
(On this step do not use any frontend framework!)
   a. Create a table that shows all the albums. It gets all the albums from the API.
   b. Each row has buttons for update, delete and show details.
      i. Update will send the new edited album details to the API which will send a put request to the server.
      ii. Delete will delete the user and update the table afterwards. It should have a confirmation dialog to confirm deletion.
   c. Create a new album. This will send the new album details to the API which will in return send a post request.

For the last step you can use **Fetch** or **Axios**.

Here is an example of how to use the Fetch API to retrieve data from your RESTful API and display it on the page:

```
fetch("http://localhost:3000/albums")
  .then(response => response.json())
  .then(data => {
    // Use the data to display albums on the page
  })
  .catch(error => {
    console.error("Error fetching albums:", error);
  });
```

GOOD LUCK