ITE5315 Project
Winter 2024

Project duration:

- Week11~13

Project submission:

- Phase 0: April 1, During the class, project planning, basic setup. Clarifications.
- Phase 1 (WK 1): April 7, Submission + April 8 class discussion. Clarifications.
- Phase 2 (WK 2): April 14, Final submission
- Phase 3: April 15, Peer Project evaluation

Assessment Weight:

- 30% of your final course Grade

Assessment includes:

- Submitting project WK 1 (posted in BB) (10%)
- Final project submission includes all project files and video (18%)
- Project peer evaluation and discussion in Week 13 (2%)

Submission Details:

- Submission Deadline: April 14th

Project Overview:

- Objective: To explore more on developing a DB-driven Node/Express app (REST API)
- Description: Using the concepts we learned during the course, we will develop a secure, DB-driven Node/Express app.

Project Specification:

WK 1 – Step 0:

- Create a folder/directory for your project. Call the folder/directory "5315-project"
- It is recommended that you use Git to manage your source code.
    - Create a new repository in GitHub called 5315-project
    - In the project folder, create a git repository ("git init .")
    - Follow the instructions in GitHub to add the repository as a remote to your new repository
    - Create a readme file, add the file to the repo and push the commit to the 5315-project repo in GitHub
    - You can install GitHub Desktop from here (if you prefer a GUI). https://desktop.github.com/
    - You can install git bash if you prefer command line from here. https://git-scm.com/downloads

WK 1 - Step 1: Loading the "Restaurant Data" in MongoDB Atlas.

- Create a new database called "5315-project". Create a collection called "restaurants".
- Using Compass import the provided restaurants.json file into the "restaurants" collection

WK 1 - Step 2: Building a Web API (9 marks)

- You need to install express and mongoose and any other modules you require.
- Add a "start" entry to the "scripts" section of your package.json. point it to your main app file. e.g. "start": "node app.js". test using "npm run start"
- Implement a module to interact with the Restaurant MongoDB collection (Similar to Assignment 4)
- "Initializing" the Module before the server starts
    - To ensure that we can indeed connect to the MongoDB Atlas cluster with our new connection string, we must invoke the db.initialize("connection string…") method and only start the server once it has succeeded, otherwise we should show the error message in the console
- This module will provide the 6 async functions required by our Web API for this dataset:

- db.initialize("Your MongoDB Connection String Goes Here"): Establish a connection to the MongoDB server and initialize the "Restaurant" model with the "restaurant" collection (used above)
- db.addNewRestaurant(data): Create a new restaurant in the collection using the object passed in the "data" parameter
- db.getAllRestaurants(page, perPage, borough): Return an array of all restaurants for a specific page (sorted by restaurant_id), given the number of items per page. For example, if page is 2 and perPage is 5, then this function would return a sorted list of restaurants (by restaurant_id), containing items 6 – 10. This will help us to deal with the large amount of data in this dataset and make paging easier to implement in the UI later. Additionally, there is an *optional* parameter "borough" that can be used to filter results by a specific "borough" value
- db.getRestaurantById(Id): Return a single restaurant object whose "_id" value matches the "Id" parameter
- updateRestaurantById(data,Id): Overwrite an existing restaurant whose "_id" value matches the "Id" parameter, using the object passed in the "data" parameter.
- deleteRestaurantById(Id): Delete an existing restaurant whose "_id" value matches the "Id" parameter

- Add the routes: The next piece that needs to be completed is to define the routes (listed Below). Note: Do not forget to return an error message if there was a problem and make use of the status codes 2xx, 4xx and 500 where applicable.
  - POST /api/restaurants
    This route uses the body of the request to add a new "Restaurant" document to the collection and return the created object / fail message to the client.

  - GET /api/restaurants
    This route must accept the numeric query parameters "page" and "perPage" as well as the string parameter "borough", ie: /api/restaurants?page=1&perPage=5&borough=Bronx. It will use these values to return all "Restaurant" objects for a specific "page" to the client and optionally filter by "borough", if provided.
    Add query param validation to your route to make sure that the params you expect are present, and of the type you expect. Use the “express-validator” module (https://express-validator.github.io/docs/) for this (or another library if you want). If the params are incorrect, your route should return a 400

response (client error) vs. 500 (server error).

- GET /api/restaurants
  This route must accept a route parameter that represents the _id of the desired restaurant object, ie: /api/restaurants/5eb3d668b31de5d588f4292e. It will use this parameter to return a specific "Restaurant" object to the client.

- PUT /api/restaurants
  This route must accept a route parameter that represents the _id of the desired restaurant object, ie: /api/restaurants/5eb3d668b31de5d588f4292e as well as read the contents of the request body. It will use these values to update a specific "Restaurant" document in the collection and return a success / fail message to the client.

- DELETE /api/restaurants
  This route must accept a route parameter that represents the _id of the desired restaurant object, ie: /api/restaurants/5eb3d668b31de5d588f4292e. It will use this value to delete a specific "Restaurant" document from the collection and return a success / fail message to the client.

WK 1 - Step 3: Add UI

- You want to demonstrate your skill in working with Template Engines and Forms, but you don't want to apply this for the entire application.
- Add a new route which works like "/api/restaurants?page=1&perPage=5&borough=Bronx", but takes the 'page', 'perPage' and 'borough' from a FORM and displays the output using Handlebars template engine. Use the same route, but with different HTTP methods (GET, POST) to determine if only to render the form or the form and the results. (4 marks)
- Use your creativity to design the layout and apply proper css style/format. (2 mark)

Submit your WK 1 current Source Code and Project Document (to-date, including screenshots) via Blackboard at the end of Week 1 (April 7th)

WK 2 - Step 4: Add security features to the app (4 marks):

- Use an Environment Variable for your Connection String: Your solution currently has your database connection string (with username and password!) hard coded into your source code. This is a potential security risk. If this code is shared between users on a team, or if it is pushed to a public repo on GitHub, your password is now public too :(
- The app gives access to restaurant data. How do you limit access, so that only authorized users can access specific routes?
- The goal is to encrypt sensitive data (i.e. passwords) and use JWT/Sessions to secure routes for authorized users (note: for the scope of this project, Users and associated data can be designed locally in app or from a DB collection – you can pick).
- You are asked to use security features like Password Encryption, JWT, Session/Cookie that we have learned in the course to allow only authorized users to use the following routes:
    a. POST /api/restaurants
    b. PUT /api/restaurants
    c. DELETE /api/restaurants

WK 2 - Step 5: Add new functionality and/or a feature to the app (3 marks)

- Use your creativity to add a new feature to the app. This can be adding a new UI/route/DB operation, or by using a new npm package or style library that enhances your design. You could add a GraphQL implementation of your API!

WK 2 - Step 6: Publishing to Vercel (2 marks)

- Once you are satisfied with your application, deploy it to Vercel. We will go over an example of publishing to Vercel in class. If you want to read-ahead you can follow the process yourself using these guides:
    - Using the Vercel CLI (does not require git):
        - first: `npm i -g vercel.`
        - Read https://vercel.com/docs/deployments/overview#vercel-cli
        - Then https://vercel.com/docs/cli/deploy
    - Using github and Vercel integration (recommended): https://vercel.com/docs/deployments/git#deploying-a-git-repository

Project expectations:

- Following best practices in handling async tasks
- Following best practices in error handling
- No use of var keyword
- Practice good in structuring of a NodeJs app and separate functionality / features
- Please use the practices and approaches that we used in other activities during this course.

(continued next page)

Project Submission:

- Add the following declaration at the top of .js files

/**

ITE5315 – Project

I declare that this assignment is my own work in accordance with Humber Academic Policy.

No part of this assignment has been copied manually or electronically from any other source

(including web sites) or distributed to other students.

Name: Student ID: Date:

**/

- Compress (.zip) the source code files in your project directory. Important: Do NOT include the node_modules directory or any .env files.
- Complete & Submit the Project Document (this will be provided in WK 2).
- Record a detailed walk-through/demonstration video presentation of the project

Important Note:

Submitted code **must** run locally, e.g.: start up errors causing the assignment/app to fail on startup will result in a grade of zero (0) for the assignment.

LATE SUBMISSIONS for assignments. There is a deduction of 25% per day for Late submissions, and after two days it will grade of zero (0).

Assignments should be submitted along with a video recording which contains a detailed walkthrough of your solution. Without recording, the assignment can get the maximum of 1/3 of the total marks.