Node.js & MongoDB: Even Better Together (Integrating Simple "Subscriber C/R/U/D" Into a Web-Application)

Avraham Leff

COM3580: Spring 2024

1 Assignment-Specific Packaging

The general packaging is unchanged from the basic "Homework Requirements" (see slides from first lecture and "Homework Policies for COM 3580" on Piazza).

This assignment's *top-level* "DIR" **must be named** *NodeAndMongo03*. You will be storing *Node.js* code in "DIR", important details are provided below. You will not need to create a writeup file, because I plan to test your implementation by pulling the code from Git and running the application with a series of steps (specified below).

Because **you must package your code** such that I can create your server, create the database, and then pass URLs to your server, you must follow the (standard) *Node.js* conventions for packaging an application (details below).

2 Motivation

A "command-line" shell offers a convenient way to perform individual interactions with a database. Often, however, implementing a "data pipeline" or building an application requires the sort of flexibility offered only by a *general-purpose* programming language.

This assignment is intended to give you experience with such a "general purpose" programming approach. We'll use *Node.js* to integrate your business logic and MongoDB data. To make things more interesting, you will package this integrated function as a web-application.

> Specifically: in this assignment, your *Node.js* application will interact with MongoDB through a JavaScript MongoDB driver.
>
> Note: you can use any asynchronous API flavor (e.g., "wait-async" or Promises or callbacks) that you choose.
>
> Also, you can choose to interact directly with MongoDB using a MongoDB Node Driver or you can use a higher-level framework such as as Mongoose.

Node.js is a very popular server platform and is the anchor of the MERN stack. (See, also this article.)

> Node.js is an open source, cross-platform JavaScript runtime environment for developing server-side and networking applications. This popular server platform is built on Chrome's V8 JavaScript engine and enables you to use JavaScript code outside of a browser

> Warning: getting set up to use *Node.js*, `npm`, *asynchronous* APIs, and the MongoDB driver may take longer than you anticipate! I urge you to get started immediately.
>
> I suggest that you consider using *Node.js* middleware frameworks such as Express to build your web-server.

3 Setting Things Up: Preliminaries

3.1 *Node.js*

1. Install the *Node.js* runtime and package manager (NPM) on your machine.

There are several ways to do this (there are pros and cons for each, I suggest that you explore a bit before picking one). Regardless of which approach you use, I must be able to build your web-app using the steps described below.

- The *Node.js* download page
- (For Mac users), can install using Homebrew

2. Verify that you're ready to get to work.

   (a) `node --version`

   - Should respond with `v21.6.1` (or approximately so)

   (b) Create a file named `hello.js`, containing this code:

```
1        console.log('Hello, Node.js!');
```

   (c) Invoke the following command: you should see the following output.

```
avraham@leff-imac:NodeAndMongo3: more hello.js
console.log('Hello, Node.js!');
avraham@leff-imac:NodeAndMongo3: node hello.js
Hello, Node.js!
```

3.2 Setting Things Up: MongoDB

You already have MongoDB installed on your laptop. Your MongoDB installation **must** use the default configuration of listening on port **27017**.

4 Requirements

Do not check in the `node-modules` directory (including all subdirectories that are generated when you install packages). Use `.gitignore` to ensure that these artifacts are not checked in! **Points will be deducted** if you don't follow these instructions.

I'm not requiring "polished" code, **certainly not** a beautiful GUI! For this assignment, focus on having the code deliver the specified function and that can be built using the following sequence of steps.

Your code **must be organized** such that I can successfully run the following steps:

1. Your code **must include** a `package.json` and `package-lock.json` so that I can do an `npm install` to build your code and **then execute** `node main.js` **to run your code**.

   You can include whatever dependencies you need in your `package.json`. The only requirement is that

   - The mandatory *mongodb* dependency be (approximately) `6.3.0`
   - The mandatory *mongoose* dependency be (approximately) `8.1.2`

     Note: you **do not have to use** Mongoose in your implementation! You **do have to include** the above dependency in your `package.json`.

   Note:

   - You will earn a "0" if I can't successfully "build" your code using the above steps.
   - You have almost complete flexibility with regard to how you structure your code. However: the following **must be located in the top-level "DIR"**.

     ```
     avraham@leff-yumacpro-2:NodeAndMongo3\$ ls -1
     main.js
     package-lock.json
     package.json
     ```

     See e.g., this explanation of the purpose of the two json files. Make sure that your `package-lock.json` is up-to-date and contains exactly what you used to build the application.

2. I plan to test your code with the following steps: you may want to use a similar check-list as you develop.

   (a) Ensure that MONGODB is running

(b) Pull your code and `cd DIR`

(c) Execute `npm install` followed by `node main.js`

(d) Your web-server must launch such that I can access your application at `localhost:3000`.

4.1 MongoDB & Application Data

- In this iteration, the database **must be named** NodeAndMongo3_db. Note the "underscore"!
- The database contains only one collection which **must be named** subscribers.
- A **subscriber** document **must** contain the following attributes named **exactly**
  - **name**, **email**, **zipCode**. The values for these attributes are all of type **String**.

  In addition, a **subscriber** document **may** contain an **age** attribute of type **Number**.
- All documents created by the web-application **must be persisted** as described above.
- The collection may contain (valid) documents that were created "out-of-band" of the web-application.
- You need not worry about out-of-band creation of invalid documents (despite this being MongoDB ☺).

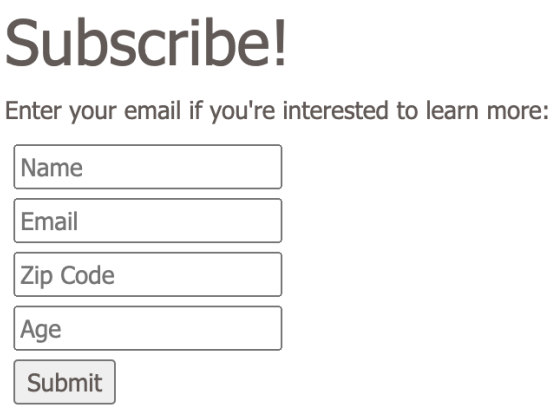
4.2 Web-Application Function

- Your application **must respond** to any valid client request (whether GET or POST) with a "200" code response.
- Your application **must respond** to any client request that isn't one of the following URLs (whether GET or POST) with a "404" code response and a corresponding "error page".

- Your application **must respond** to any client request that triggers a server-side error with a "500" code response and a corresponding "error page".
- Your application **must respond** to any client request that contains invalid arguments with a "400" code response and a corresponding "error page".
- Top-level application page, reachable via `localhost:3000`. At a minimum, must display *NodeAndMongo3 Web-Application* and the following "clickable" buttons.
  - *Home*: clicking this button should bring up the "home page" at URL `http://localhost:3000/`.
  - *Contact*: clicking this button should bring up the "add subscriber" page at URL `http://localhost:3000/contact`.
  - *Subscribers*: clicking this button should bring up the "list of subscribers" page at URL `http://localhost:3000/subscribers`.
  - *AgeFilters*: clicking this button should bring up the "filter by age" page at URL `http://localhost:3000/age-filter`.

4.2.1 http://localhost:3000/contact

- The "add subscriber" page brings up a form resembling the following (I don't care about the "looks", but the detailed function **does matter**: e.g., **input field names** and form **actions**).



The form should contain **exactly these input fields**:

  - A text input field named name. Return a "400" and an error page if the input is not at least one character long.
  - A text input field named email. Return a "400" and an error page if the input is not at least one character long.
  - A text input field named zipCode. Return a "400" and an error page if the input is not at least one character long.
  - A **number** input field named age. Valid inputs must be > 0 and ≤ 100. With respect to processing the form, this field is **optional** (the client need not supply her age).
  - An input field of type submit, named submit.

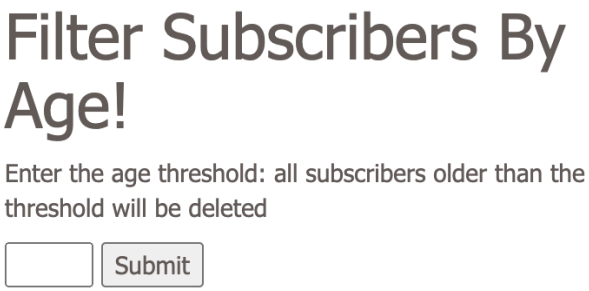
- Given valid input, clicking on the submit button must result in creating a *subscriber* document, and adding it to the MongoDB collection (details above).
- The form **must be** submitted as a POST request to URL `/subscribe`.

4.2.2 http://localhost:3000/subscribers

- GET `http://localhost:3000/subscribers` should return a formatted list of all the subscribers in the collection.
  - Do not display any properties other than name, email, zipCode, and age (if present).
  - For this iteration, you may assume that all documents will have the mandated properties.
  - If a document is missing the `age` property, **you must** omit that property from the displayed document (but display the other, mandatory, properties)

4.2.3 http://localhost:3000/age-filter

The "age filter" page brings up a form resembling the following (I don't care about the "looks", the detailed function does matter: e.g., **input field names** and form **actions**).



The form should contain **exactly these input fields**:

- A **number** input field named agefilter. Valid inputs must be > 0 and ≤ 100.
- An input field of type submit, named submit.

Given valid input, clicking on the submit button must delete all *subscriber* documents whose age is $\geq$ the supplied age.

- The form **must be** submitted as a POST request to URL `/process-age-filter`.
- The filter has no effect if a document doesn't include a document property.

5 Grading

- Your web-application must build and launch successfully and then process client requests at port **3000**. Failure to do so is an automatic "0".
- You will be graded based on the degree to which you supply the function specified above (pay attention to the requirements details!).
- If you want, you can create a writeup file containing good quality screenshots of the web-browser handling your test case input.

  The writeup file is only a **"(very) partial credit" backup** in case your implementation doesn't launch properly.

Your web-application must respond to a client request within half a second!

5.1 Dry Run

Some of you may not be familiar with *Node.js* packaging, so I plan to do a "dry run" several days before the assignment is due. This is not mandatory: if you wish, let me know, and I'll pull your code and see if I can build and launch your web-server successfully. I won't report if the code is "correct", I'll only give feedback as to whether the automated build process succeeded.