CSE186 Summer 2024

Assignment 6

In this assignment you will write a Node.js & Express RESTful E-Mail API backed by a PostgreSQL database and associated tests to demonstrate it works as required.

This assignment is worth 10% of your final grade.

Late submission will not be accepted.

Installation

See instructions in Assignment 1 and ensure you are running the LTS version of Node.js.

You will also need to download and install Docker Desktop:

https://www.docker.com/products/docker-desktop

Setup

Download the starter code archive from Canvas and expand into an empty folder. I recommend, if you have not already done so, creating a folder for the class and individual folders beneath that for each assignment.

The starter code archive contains three files that you will modify:

```
api/openapi.yaml
src/app.js
test/basic.test.js
test/advanced.test.js
```

Do not modify any other file.

Note you will add at least one JavaScript file to `src`.

To setup the development environment, ***navigate to the folder where you extracted the starter code*** and run the following command:

```
$ npm install
```

This will take some time to execute as `npm` downloads and installs all the packages required to build the assignment. You can ignore any warnings.

Now start the development database:

```
$ docker-compose up -d
```

The first time this runs it will take a while to download and install the Docker PostgreSQL image and start the database service for your server to run against.

To stop the development database: **(do not do this now, do it when you've completed the assignment)**

```
$ docker-compose down
```

To start the dev server, run the following command:


```
$ npm start
```

To execute tests run the following command:

```
$ npm test
```

To run the linter against your code, run the following command:

```
$ npm run lint
```

Background

E-Mails in this system have the following properties:

- `id` Universally Unique Identifier https://www.uuidtools.com/what-is-uuid
- `from` See below
- `to` See below
- `subject` String
- `sent` ISO Format Date String
- `received` ISO Format Date String
- `content` String

Where `form` and `to` have the following properties:

- `name` String
- `email` E-Mail Address

TAKE CARE: THIS IS NOT THE SAME AS ASSIGNMENT 5

Note that your API must be constrained by an OpenAPI version 3.0.3 schema. You should endeavor to make this schema as restrictive as possible.

For example, your schema should specify the acceptable format of the e-mail id property. If an id in any other form is presented, your system should reject it simply by performing the validation provided in the starter code.

When you start the server, use a browser to visit http://localhost:3010/v0/api-docs/ where you can manually test your API as demonstrated in class for the books example.

Initially, the only operation available will be:

Which will not work until you implement the necessary Express route.

Once that is working, you will need to add additional endpoints in the OpenAPI schema and additional Express routes to service them.

If you have no idea what any of those terms represent, view the recording of lecture 12 “Node.js & Express”, and study the handouts from the same lecture before consulting with the TAs and/or Instructor.

Resetting Docker

If you run into problems with your dev database, or simply want to re-set it to its initial state, issue the following commands to shut docker down:

```
$ docker stop $(docker ps -aq)
$ docker rm $(docker ps -aq)
```

Then restart the development database:

```
$ docker-compose up -d
```

Database Schema

The supplied database schema can be found in `sql/schema.sql`. It defines two tables '`mailbox`' and '`mail`' study the schema for the join between them.

Requirements

Basic:

- Write an OpenAPI compliant E-Mail RESTful API, backed by the PostgreSQL database, with the following informally described endpoints:
  - **GET /v0/mail**

    Return all the emails in the system as an array of named mailboxes containing arrays of e-mails, one per mailbox. For example:

    ```
    [
      {"name": "inbox", "mail": [<e-mails in inbox>]},
      {"name": "sent", "mail": [<e-mails in sent>]},
      {"name": "trash", "mail": [<e-mails in trash>]},
      ...
    ]
    ```

    Note that all returned e-mails should have had their 'content' property removed.

  - **GET /v0/mail?mailbox={mailbox}**

    Return all the emails in a specified **mailbox** as an array containing one named mailbox containing an array of e-mails in the mailbox. Throw a 404 error if the mailbox is unknown. For example, if the query parameter "mailbox" was equal to 'trash", return:

    ```
    [ {"name": "trash", "mail": [<e-mails in trash>]} ]
    ```

    Note that the returned e-mails should have had their 'content' property removed.

  - **GET /v0/mail/{id}**

    Return the e-mail identified by **id**, including its 'content' property. Throw a 404 error if **id** does not identify a known e-mail.

  - **POST /v0/mail**

    Save a new email sent in the HTTP request body into the sent mailbox and return the newly created e-mail. The `id`, and from properties should be missing on submission and set on return; `id` should be set to a valid UUID, `from.email` and `from.name` should be as for all existing emails in the sent

mailbox. The `sent` and `received` properties should also be missing from the submitted email with `sent` being set appropriately on return with `received` holding the same value. Throw a 400 error if the submitted e-mail has any unexpected properties.

- **PUT /v0/mail/{id}?mailbox={mailbox}**

  Move the e-mail identified by **id** into the named **mailbox**. If the mailbox does not exist, create it and move the identified e-mail into it. Return an HTTP 204 on success and throw a 404 error if **id** does not identify a known e-mail. Throw a 409 error if the named mailbox is 'sent' and the mail identified by **id** is not already in the sent mailbox.

Advanced:

- Write tests to demonstrate your implementation satisfies the Basic Requirements.
- Show your tests achieve 100% line, statement, branch, and function code coverage.
- Demonstrate zero lint errors, warnings or suppressions in your code.

Stretch:

- Modify your OpenAPI schema to include the following informally described endpoint:
  - **GET /v0/mail?from={from}**

    Returns an array of named mailboxes containing arrays of e-mails sent by the individual identified by **from** where **from** can be a full or partial name like 'jOhn sMiTh' or 'JoN' or a correctly formatted email address like johnny@bigco.com. In both cases, **from** is case *insensitive*. Note that the returned e-mails should have had their 'content' property removed. Return HTTP 200 if at least one mail is found, 404 if not.

- Write tests to demonstrate this new end point works.
- Continue to show zero lint problems and 100% code coverage.

What steps should I take to tackle this?

You should base your implementation on the Books Database Example from class. Start by implementing the simple GET endpoints then move on to the more complex POST and PUT operations.

There is a huge amount of information available on OpenAPI 3.0.3 at https://swagger.io/specification/ and http://spec.openapis.org/oas/v3.0.3. Also consult the Petstore example at https://petstore3.swagger.io/ and make use of the on-line schema validator at https://editor.swagger.io/.

A good resource for querying JSON stored in PostgreSQL: https://www.postgresqltutorial.com/postgresql-json/

How much code will I need to write?

Much of this implementation will be in the OpenAPI schema in `api/openapi.yaml`. if you find yourself writing "defensively" in your JavaScript to check for incorrect data sent by the caller, you should instead make your OpenAPI schema more robust such that it can be used to reject invalid submissions.

Assuming you OpenAPI schema is sufficiently robust, your solution is likely to contain somewhere in the region of 200 lines of YAML and 150 lines of JavaScript.

Grading scheme

The following aspects will be assessed:

1. (100%) **Does it work?**

    a. Basic (50%)
    b. Advanced (30%)
        i. 50% deduction for any lint errors, warnings, suppressions
        ii. 50% deduction for < 100% code coverage
    c. Stretch (20%)
        i. 50% deduction for any lint errors, warnings, suppressions
        ii. 50% deduction for < 100% code coverage

2. (-100%) **Did you give credit where credit is due?**

    a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%). You will also be subject to the university academic misconduct procedure as stated in the class academic integrity policy.

    b. Your submission is determined to be a copy of a past or present student's submission (-100%)

    c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

        - < 25% copied code — No deduction
        - 25% to 50% copied code — (-50%)
        - > 50% copied code — (-100%)

What to submit

Run the following command to create the submission archive:

```
$ npm run zip
```

**** UPLOAD** `CSE186.Assignment6.Submission.zip` **TO THE CANVAS ASSIGNMENT AND SUBMIT ****

