CSE187 Spring 2024

Assignment 1

In this assignment you will write a Facebook™ style TypeScript, TSOA, Node.js & Express OpenAPI Compliant RESTful 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

Ensure you are running the current LTS version of Node.js and have Docker Desktop installed.

Setup

Download the starter code archive from Canvas and expand into an empty folder. I recommend creating a folder for the class and individual folders beneath that for each assignment.

You must not modify any files in the starter code, with one exception:

```
src/types/index.d.ts
```

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:

```
$ docker-compose down
```

To start the dev server, run the following command:

```
$ npm run dev
```

To execute tests run the following command:

```
$ npm test
```

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

```
$ npm run lint
```


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

Define your schema by editing `sql/schema.sql` (without removing the existing member table definition) and create dev data by writing insert statements for it in `sql/data.sql`. Note the test database contains a single member with administration rights and no other data; **do not modify** `sql/test.sql`.

Background

Facebook, in its simplest form, has just two data entities:

Member	User account
Post	Owned by Members who grant access to other Members known as “Friends”

Yet the concept of friendship between members introduces the following meta-data:

Friend	Member allowed to see another Member’s Posts
Request	One Member asking another if they would like to be their Friend By accepting a Request, the accepter is allowing the requestor to see all their posts

When Member A wants to be friends with Member B:

- A sends B a Request
- If B accepts the Request, A & B are now Friends
- If B rejects the Request (or simply never accepts it) A & B are not Friends

Only when B accepts A’s Request can A see B’s Posts and B can see A’s Posts.

API Definition

For a complete implementation you'll need the following informally described endpoints.

All return a Code of **401 Unauthorised** in appropriate circumstances. Note that admin users can only login and create new Members.

Unauthenticated:

- **POST /api/v0/login**

  Accepts: `{email: string, password: string}`
  Returns: `{id: uuid, name: string, accessToken: JWT}`
  Codes: 200 OK

Authenticated:

- **POST /api/v0/member**

  Description: Create a new Member - Administrators only (Anna Admin, not Molly Member)
  Accepts: `{email: string, password: string, name: string}`
  Returns: `{id: uuid, name: string}`
  Codes: 201 Created, 409 Conflict

- **GET /api/v0/member**

  Description: All Members
  Returns: `[{id: uuid, name: string}]`
  Codes: 200 OK

- **GET /api/v0/request**

  Description: Members who have requested the logged in Member be their Friend
  Returns: `[{id: uuid, name: string}]`
  Codes: 200 OK

- **PUT /api/v0/request/{memberId}**

  Description: Accepts the Friend Request from Member with id `memberId`
  Returns: `{id: uuid, name: string}` The Member who made the request
  Codes: 200 OK, 404 Unknown

- **GET /api/v0/friend**

  Description: Members who are, or who have been requested to be Friends of the logged in Member
  Returns: `[{id: uuid, name: string, accepted: boolean}]`
  Codes: 200 OK

- **POST /api/v0/friend{memberId}**

  Description: Requested Member with id `memberId` become a Friend of the logged in Member
  Returns: `{id: uuid, name: string, accepted: false}`
  Codes: 200 OK, 404 Unknown, 409 Conflict

- **DELETE /api/v0/friend{memberId}**

  Description: Remove friendship (if any) between `memberId` and the logged in Member
  Returns: `{id: uuid, name: string, accepted: boolean}`
  Codes: 200 OK, 404 Unknown

- **POST /api/v0/post**

  Accepts: `{content: string, image: url}`
  Returns: `{id: uuid, member: uuid, posted: datetime, content: string, image: url}`
  Codes: 201 Created

- **GET /api/v0/post?page=number**

  Description: All posts created by the logged in Member and any of their Friends in datetime order, most recent first, in pages of 20 starting at page `number`, a positive nonzero integer.
  Returns: `[{id: uuid, member: uuid, posted: datetime, content: string, image: url}]`
  Codes: 200 OK

Requirements

For a complete implementation you'll need the endpoints described above, but not all of them and not necessarily all their features are required for Basic so "**code to the test**" implementing only what you need to pass. You'll worry about all the niceties when working to satisfy the Advanced Requirement

Basic:

- Pass the supplied basic tests: `npm test basic`
- Show your tests achieve 100% line, statement, branch, and function code coverage.
- Demonstrate zero lint errors or warnings in your code.

Advanced:

- Write tests to demonstrate your API meets ***all*** the requirements defined above, not simply the ones required to pass the basic tests.
- Continue to demonstrate 100% line, statement, branch, and function code coverage.
- Maintain zero lint errors or warnings in your code.

What steps should I take to tackle this?

Remind yourself of the work you did for the later CSE186 Assignments, especially the Mini Project, and take inspiration from the **TypeScript Database Book Example** and **TypeScript Authenticated Book Example** covered in class.

We'll be covering numerous issues in Secret Sauce but make sure to run your ideas past your classmates, TA and instructor before spending a lot of time on something you're not entirely sure about.

As always, take small steps. Initially, you need the login endpoint so the given administrative user can authenticate themselves. Next, you'll need the ability to POST new Members, then POST and GET Posts, and so on.

It is highly recommended that you tackle the required endpoints in the order they're needed in the supplied basic test suite. As you add more endpoints, you'll pass more tests.

Note that you cannot pre-populate the test database with new members, posts and so on. You have to "**use the API to test the API**", an important skill to develop.

Grading scheme

The following aspects will be assessed:

1. (60%) **Basic Requirement?**

    a. Pass Basic Tests (40%)
    b. Perfect line, statement, branch, and function code coverage (15%)
    c. Zero lint errors or warnings (5%)

2. (40%) **Advanced Requirement?**

    a. Pass known-good tests on all endpoints (40%)

3. (-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** `CSE187.Assignment1.Submission.zip` **TO THE CANVAS ASSIGNMENT AND SUBMIT ****

