**CSE186 Summer 2024**

**Assignment 7**

In this assignment you will write a simple Full Stack Web App and 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 current LTS version of Node.js. You will also need to have downloaded and installed 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.

**Do not modify any configuration files in the starter code or any source file that contains a comment instructing you not to do so.**

Note only the contents of `backend/src`, `backend/api` and `frontend/src` are included in your submission.

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.

To start the database Docker container, in the `backend` folder run:

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

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

To start the frontend and backend dev servers, run the following command:

```
$ npm start
```

The frontend and backend servers can be run individually from their respective folders by running `npm start` in separate console / terminal windows.

To run the linter against your API or UI code, run the following command in the `backend` or `frontend` folder:

```
$ npm run lint
```

To fix some linter errors, run the following command in the `backend` or `frontend` folder:

```
$ npm run lint -- --fix
```

 Copyright © 2024 David C. Harrison. All rights reserved.

To run API tests run the following command in the `backend` folder:

```
$ npm test
```

To run UI tests run the following command in the `frontend` folder:

```
$ npm test
```

To run end-to-end tests run the following command in the `e2e` folder:

```
$ npm test
```

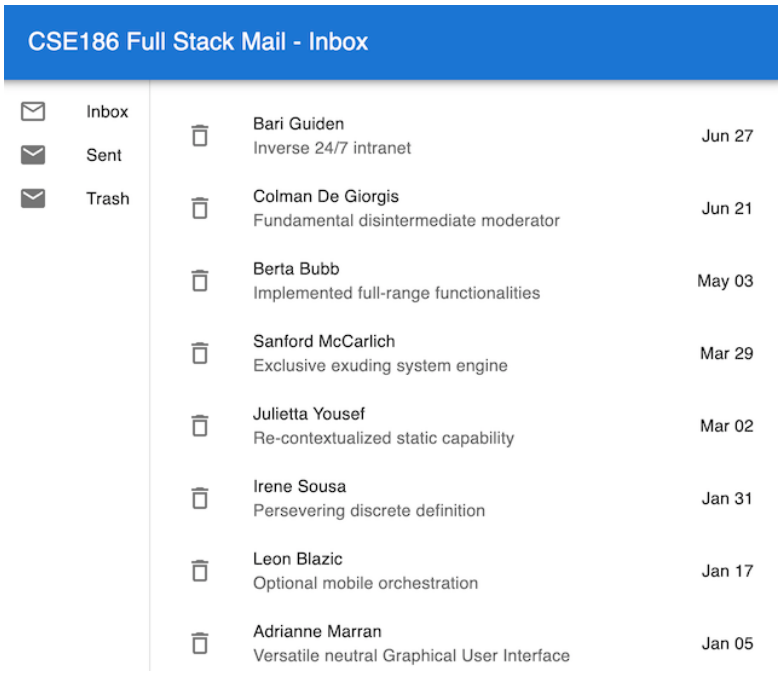
To stop the database, run the following command in the `backend` folder

```
$ docker compose down
```

## Background

This assignment combines a simple version of Assignment 4 and a simple version of Assignment 6 into a full stack Web App. However, it is not recommended you take your solutions to Assignment 4 and 6 and try to modify them, instead, start from scratch and bring code in as needed.

Your UI might end up looking something like this:




## 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
```

 Copyright © 2024 David C. Harrison. All rights reserved.

## Database Schema

The supplied database schema can be found in `backend/sql/schema.sql`. It defines two tables '`mailbox`' and '`mail`' with a join between them.

Note this database schema, whilst similar, is not the same as the one in Assignment 6.

## Requirements

Basic:

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

      Return the names of all mailboxes known to the system as an array of strings.
    - **GET /v0/mail?mailbox={mailbox}**

      Return all the emails in a specified **mailbox** as an array. Throw a 404 error if the mailbox is unknown. Note that the returned e-mails should have had their 'content' property removed.
    - **PUT /v0/mail/{id}?mailbox={mailbox}**

      Move the e-mail identified by **id** into the named **mailbox**. If the mail or the mailbox does not exist, throw a 404 error. Throw a 403 error if the named mailbox is 'sent'. Return 204 on a successful move.
- Write tests to demonstrate your API 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.

Advanced:

- Write a React/MUI User Interface with the features listed below. Note that you are aiming to pass the supplied e2e tests (see below) so study them for required aria-labels and text.
    - Show the contents of inbox at startup
    - Allow selection of other mailboxes
    - Allow "deletion" of emails in mailboxes other than trash - deletion moves the email in question to the trash mailbox and removes it from the one it was in
    - Do not allow deletion of emails from the trash mailbox
- Write tests to demonstrate your UI satisfies the Advanced Requirements.
- Show your tests achieve 100% line, statement, branch, and function code coverage.

Stretch:

- Pass the supplied e2e (end-to-end) tests

 Copyright © 2024 David C. Harrison. All rights reserved.

## What steps should I take to tackle this?

Develop in vertical 'slices' through the Web App, one piece of functionality at a time.

A sensible approach would be to tackle the mailboxes first:

1. Write test(s) for the /v0/mailbox endpoint
2. Implement the endpoint so it passes the test(s)
3. Write test(s) from the UI component that will show the list of mailboxes using a mock API
4. Implement that UI component so it passes the test(s)
5. Run your dev UI against the real API to confirm it looks how you want

Then do the same again for the next piece of functionality, listing the contents of a mailbox.

## How much code will I need to write?

You've done everything you need to do in the assignment before, so much of the code you'll need will come from previous assignments. Having said that, and assuming you OpenAPI schema is sufficiently robust, your API is likely to contain somewhere in the region of 140 lines of YAML and 100 lines of JavaScript. Your UI should contain something in the region of 250 lines of JSX.

## Grading scheme

The following aspects will be assessed:

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

    a. Basic (known good tests are run against your code) ( 30% )

        50% deduction for any lint errors, warnings, suppressions
        50% deduction for < 100% code coverage when running your tests

    b. Advanced (just your tests) ( 30% )

        Deduction for code coverage deficiencies:

        | | |
        |---|---|
        | 100% | ( 0% ) |
        | >= 98% | ( 25% ) |
        | >= 95% | ( 50% ) |
        | < 95% | ( 100% ) |

    c. Stretch ( 40% )

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%)

 Copyright © 2024 David C. Harrison. All rights reserved.

## What to submit

Run the following command to create the submission archive:

```
$ npm run zip
```

**** UPLOAD** `CSE186.Assignment7.Submission.zip` **TO THE CANVAS ASSIGNMENT AND SUBMIT ****

 Copyright © 2024 David C. Harrison. All rights reserved.