111-1-Web-Hackathon#2 -- Hugo Eat

Prerequisite

1. If you have not done preREADME yet, please check it out first.
    - Note: There is a typo in preREADME => there should not be a ',' after "backend" command:

      ```
      "scripts": {
        "frontend": "cd ./frontend && yarn start",
        "backend": "cd ./backend && yarn server"
      },
      ```

    - Note: Please add "react-router-dom" and "react-icons" to your frontend package:

      ```
      yarn add axios react-rating-stars-component react-icons react-router-dom
      ```

    - Note: Please add "nodemon" to your backend package:

      ```
      yarn add cors dotenv express mongoose nodemon
      ```

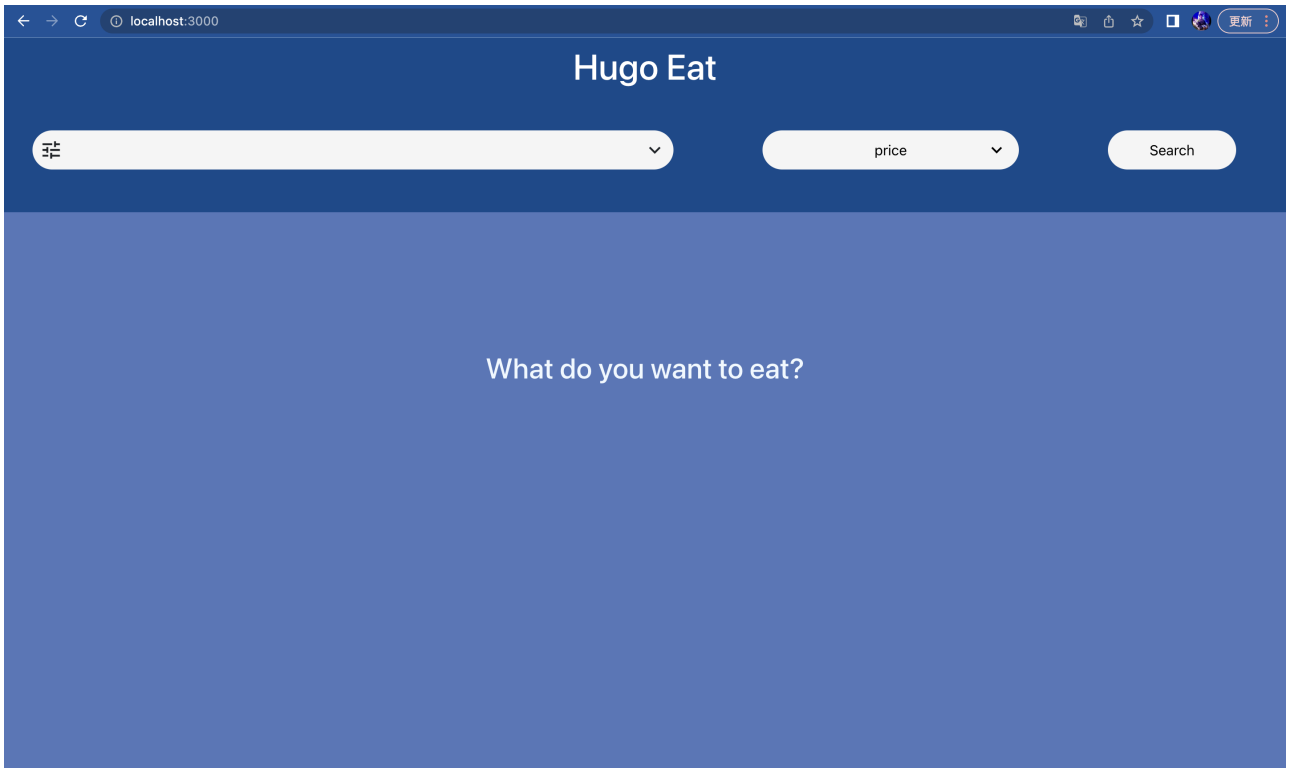
2. This in-class Hackathon is to implement a simple food rating website with basic search and commenting functionalities.
3. Please sign in to Hackathon #2 through this Google Form. After you fill in your student ID, name, department and grade, you will see the link to the reference code. **YOU NEED TO SUBMIT THE FORM** in order to complete the sign-in. Failure to sign in to the Hackathon will result in deduction in points
4. The downloaded file should be named "**hack2.zip**". Uncompress it under `hack2/` and check if the structure is identical to the figure shown in 5. (Note: If you see an extra `hack2/` directory under `hack2/`, copy the files under `hack2/hack2/ to hack2/` instead.)
5. Check the structure of the files/directories under `hack2` and it should be like this:

6. Start the frontend and backend under `hack2/` by, respectively,

   ```
   yarn frontend  # sets up frontend on localhost:3000
   ```

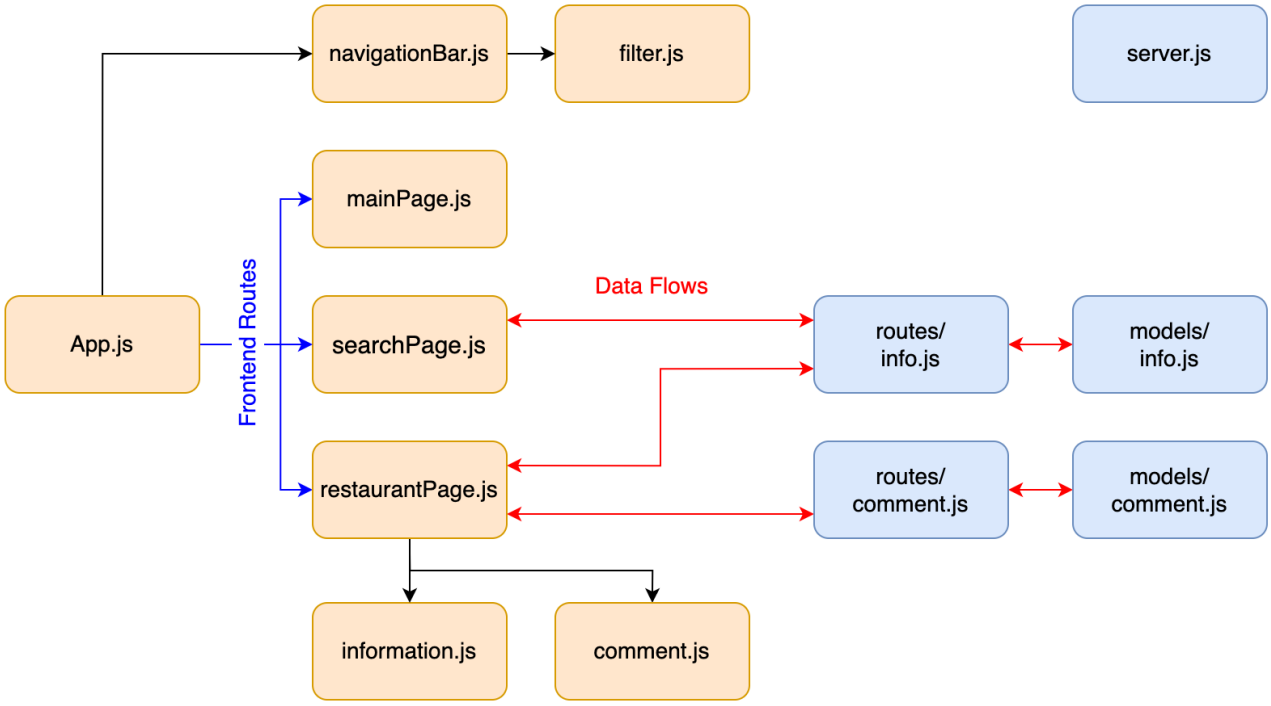
```
yarn backend # sets up backend on localhost:4000
```

7. Your backend should crash at this moment because you haven't implemented the DB connection yet. However, the initial UI for the frontend should be like this:



Reference Video (Complete version of Hugo Eat)

Code Structure



Environment

Database

- To connect to MongoDB, please specify `MONGO_URL` in `.env`. Do not hard-code it in `backend/src/server.js`.
- `MODE=` in `.env`
  - Set `MODE=Reset` in `.env` in the beginning of the execution, and the server will POST a request to reset the database to the default data.
  - In the subsequence, turn off `MODE` (i.e. unset it to `MODE=` in `.env`) in order to retain the database from the previous execution.
  - Run Cypress: You should **always** set `MODE=Reset` in `.env`
- When grading on Gradescope, we will build a local Mongo DB on the grading machine and will not use your Mongo Atlas DB cluster. The environment setups such as `.env` and `package.json` will be replaced by our files. We will test your codes by some private testcases on Gradescope.

Public Cypress tests (Optional)

- Note: Public tests provided in reference program and the private testcases on Gradescope are different. However, they are testing the same functionalities/features.
- Run cypress if your Gradescope grading is not working, or if you want to know the individual testing results of the public testcases. This is optional and has nothing to do with your final score.
- To run cypress testing, make sure you are in `hack2/` and first set

  ```
  MODE=Reset
  ```

  in `.env` to reset the database. Then, ***restart the backend server*** before running

  ```
  npx cypress run
  ```

  to run the public cypress tests.

TODOs

Please submit your code to gradescope whenever you finish a feature. See Gradescope Submission Guidelines. DO NOT hold the submission until the last minute.

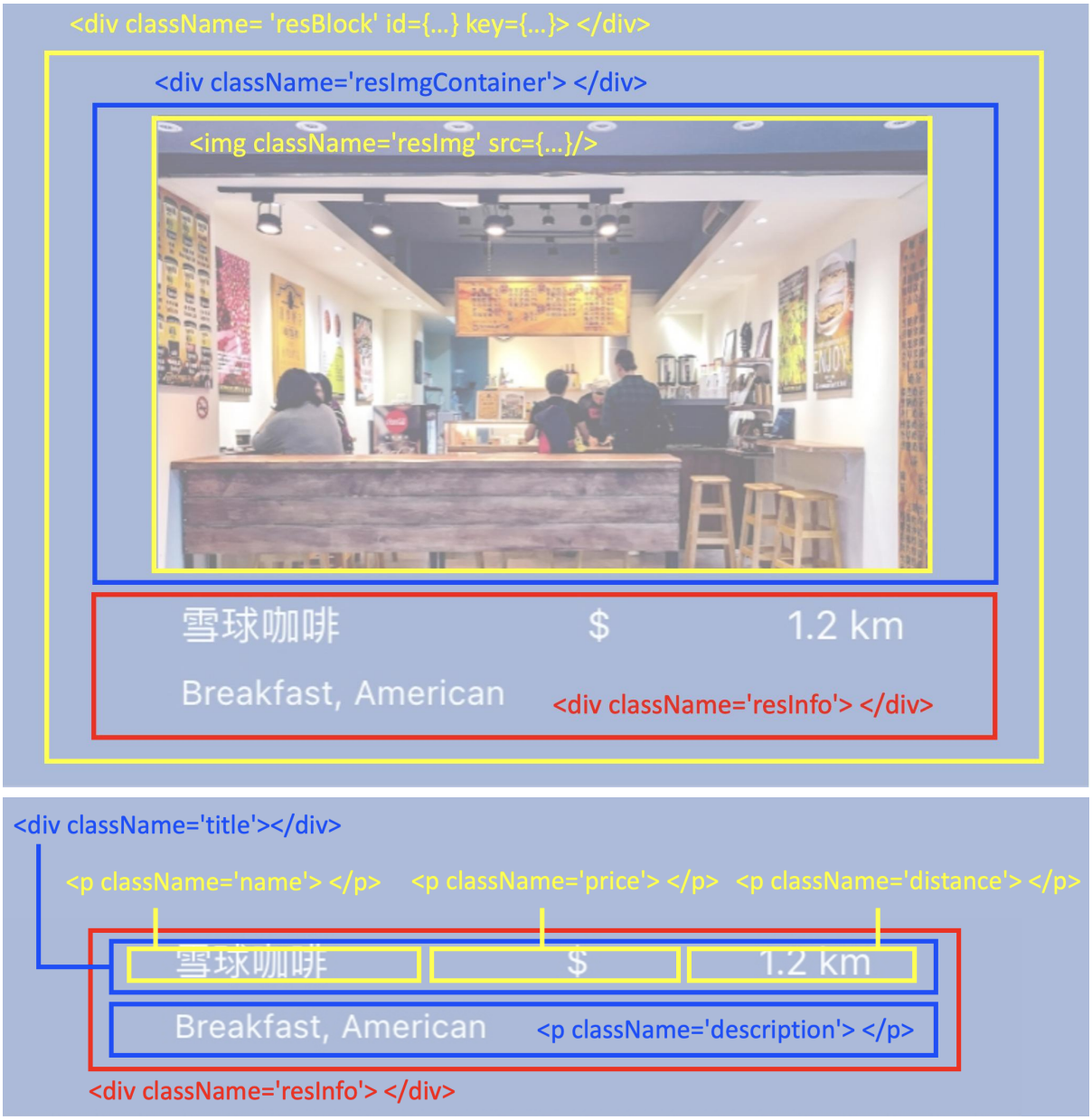
Part I - Basic Search (30%):

1. (5%) Implement Routing for `Search` Button
   - In `frontend/src/components/navigationBar.js`, use the hook `navigate` to navigate the user to the search page when clicking the Search button. You might want to refer to the routing in `frontend/src/App.js`.

- Hint:
  - Tutorial 1: Pass Data for useNavigate()
  - Tutorial 2: Redirect in React Router

2. (10%) Implement UI for Search Result (restaurants)

   - In `frontend/src/container/searchPage.js`, implement the <resBlock> ***EXACTLY AS SPECIFIED below***:
   - That is, for each search result `'item'` in the array `'restaurants'`, implement its UI by JSX as specified by the DOM structure below. Pass in the parameters `'id'`, `'img'`, `'name'`, and `'price'` of `'item'`.
   - `id` and `key` in `className='searchBlock'` should be the `id` of the restaurant
   - Note: In <description> `', '` should be added between tags. E.g. `'Breakfast, American'`.
   - Note: In <price>, 1 should be changed to `'$'`, 2 to `'$$'`, and 3 to `'$$$'`. (ref: `getPrice()`)



3. (15%) Retrieve Restaurant Information from database

- First, in `backend/src/server.js`, connect the backend to mongoDB. Please use `.env` for this task, and **DO NOT** hard-code the MONGO_URL into `server.js`. Otherwise, you may lose most of points on Gradescope.
- After setting up the connection, you can set `MODE=Reset` in `.env` and restart the backend server to initialize your database. You may clear the value of the `MODE=` tag in subsequent runs if you want to retain the data in DB and are not trying to reset the database to default.
- (10%) Then, in `backend/src/routes/info.js` implement the model `GetSearch` in order to GET the information of the restaurants.
  - Hint: use `db.collection.find({condition}).exec(err, data) {...}`
  - When success, do `res.status(200).send({ message: 'success', contents: ... })`
  - When fail, do `res.status(403).send({ message: 'error', contents: ... })`
- (5%) Finally, in `frontend/src/container/searchPage.js`, revise the function `getRestaurant()` to receive the restaurant information from the DB model `Restaurant`.
  - Hint:
    - Pass `{ params: ... }` to `axios.get()`
    - After navigation, the data you passed using navigate is accessible inside the matching route's rendered component using the `useLocation()` hook.
    - The `'location.state'` property is a user-supplied state object that is associated with this location.

Part II - Advanced Search (25%):

```
Prerequisite: Part I

This part is independent to Part III. You may choose to work on either
part first.
```

1. (5%) Handle Price/Meal/Type Filters
   - In `frontend/src/components/filter.js`, change the functions `modityFilter` so that the states `priceFilter`, `mealFilter`, and `typeFilter` change on clicks
2. (20%) Implement the `Search` function
   - Modify the Search function to support sorting and filtering.
   - Please refer to Part I for the files you might need to change:
     - (TODO) `backend/src/routes/info.js`
     - (Ref) `frontend/src/container/searchPage.js`
   - (8%) The result should be sorted according to the state `selected`, which corresponds to the sort methods, either
     - `Price (low to high)` or
     - `Distance (near to far)`

- (8%) The result should be filtered according to the state `filter`. There are three categories of filters:

```
Prices    : $        , $$      , $$$
Mealtimes : Breakfast, Lunch   , Dinner
Types     : Chinese  , American, Italian,
            Japanese , Korean  , Thai
```

  - For a restaurant to be displayed, it must contain at least one tag from each category with at least one checked filter.
  - Some examples:

```
No filter:
display all restaurants

(Breakfast, Lunch);
display restaurants with tags 'Breakfast' OR 'Lunch'

(Dinner, $$, American):
display restaurants with tags 'Dinner' AND '$$' AND 'Lunch'

(Lunch, Dinner, Japanese, Korean):
display restaurants with tags ('Lunch' OR 'Dinner') AND
('Japanese' OR 'Korean')
```

- (4%) The two functionalities (i.e. sorting and filtering) should work together. We have private testcases on Gradescope where the search result is required to be **both sorted AND filtered**.

Part III - Restaurant Page (45%)

```
Prerequisite: Part I

This part is independent to Part II. You may choose to work on either part
first.
```

1. (5%) Navigate to the Restaurant Page:
   - In `frontend/src/containers/searchPage.js`, use the hook `navigate` to navigate the user to the restaurant page when clicking a restaurant block. You might want to refer to the routing in `frontend/src/App.js`.
2. (15%) Restaurant Information

- Implement the restaurant page ***EXACTLY AS SPECIFIED below***:

  - To get the information for a restaurant, in `backend/server/routes/info.js`, implement routes to get the information of a restaurant by its `id`.
  - Then, in `frontend/src/container/restaurantPage.js`, implement the function `getInfo` to get the information of the restaurant by its id.
- (5%) **[Tags]** In `frontend/src/container/information.js`, render the tags for the restaurant. For price tags, render `$` for `price: 1`, `$$` for `price: 2`, and `$$$` for `price: 3`.
- (5%, advanced) **[Average Rating]** In `frontend/src/container/restaurantPage.js`, calculate the average rating of the restaurant, and pass it into `<Information>` via attribute `rating`.
- (5%, advanced) **[Business Time]** In `frontend/src/container/information.js`, render business time for each day of the week.
  - If the field `time` contains a single `All: "..."` tag, render the same opening time for all 7 days of the week.
  - Instead, the field `time` should contains up to 7 key-value pairs, where the keys may be `'Mon', 'Tue', 'Wed', 'Thr', 'Fri', 'Sat'` or `'Sun'`, and the value is a string containing the opening time (e.g. `{ "Mon": "07:00-24:00" }`).
  - If the restaurant does not open on a day, the corresponding key (e.g. "Sat") should be missing from `time`. In this case, please render `Closed` as the value for the opening time.

3. (25%) Comment Section

- In `backend/server/routes/comment.js`, implement routes so that it finds all comments to a restaurant.
- Then, in `frontend/src/container/restaurantPage.js`, implement the function `getComments` to get the comments of the restaurant by its `'id'`.

- (10%) **[Comment display]** In `frontend/src/container/restaurantPage.js`, implement the function `getComments` to get the comments associated with the restaurant.
- In `backend/server/routes/comment.js`, implement routes so that it creates a new comment to a restaurant.
- (10%) **[Comment writing]** In `frontend/src/container/comment.js`,
  - implement the function `storeComment` to store a new comment to the database.
  - implement the function `submitComment` to submit a comment on clicking the submit button if both name field and content field are not empty, and a rating is given (i.e., not 0). After the submission, the new comment should be stored to the database, and displayed after reloading the page. Also, the name field, content field and rating should be cleared.
- (5%, advanced) **[Automatic comment updating]** In `frontend/src/container/restaurantPage.js`, when adding new comments, update the comment display immediately, i.e., the user does not need to reload the page.

Submission

Gradescope

Please **zip your source code folders and submit them to Gradescope** by the following command.

```
// Under 'hack2/'
zip -r submission.zip ./frontend/src ./backend/src
```

The grading on Gradescope will be the final score of your Hack#2.

Please note that by default your score will be the latest uploaded 'submission.zip', **NOT the highest one**. Please be careful NOT to submit a buggy version in the last minute. Nevertheless, you can specify a specific submission for your score on Gradescope.

Push to Github

Please push your final version of codes onto the **main** branch of your GitHub repo before the end of Hack#2. We will check your codes if needed. Failure to push the code before the deadline will be treated as “no-submission” and your grade will be ***0 point***.

```
// Under 'hack2/'
git add .
git commit -m "<your commit message>"
git push
```

Should you have any questions:

Go to this Hackathon #2 Q&As repo. Check if your question has been asked. If not, open a new issue.

Please read this README.md before you do so.

Honor System

Please DO NOT discuss or consult with anyone during the exam. If you find someone who might violate this rule, or is cheating in the Hackathon, please help fill in this Google Form.