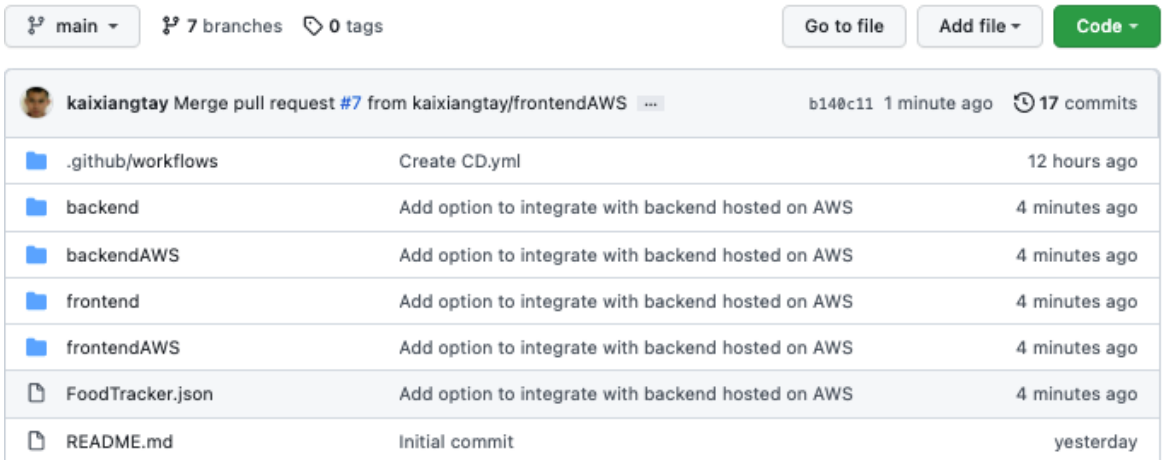
Student Name: Tay Kai Xiang Matriculation Number: A0200236Y

Link to GitHub repository: https://github.com/kaixiangtay/CS3219_OTOT_B

Pre-requisites (Software to be installed on local device):
Postman : https://www.postman.com/downloads/
NodeJS and npm: https://docs.npmjs.com/downloading-and-installing-node-js-and-npm
MongoDB: https://docs.mongodb.com/manual/installation/
(optional) MongoDB Compass: https://www.mongodb.com/try/download/compass
MongoDB Atlas account: https://www.mongodb.com/cloud/atlas/register
AWS free tier account: https://aws.amazon.com/

Step 1: Clone the project repository into the desired directory of local device.



Note: In this repository, we have 5 main folders and a file:

1) backend: local API crud implementation for Task B1 as well as local testing for Task B2

2) backendAWS : API deployment on serverless service for Task B1 and B3

3) frontend : building a Single Page Application (SPA) for Task B4 and interacting with backend on local endpoint

4) frontendAWS: building a Single Page Application (SPA) for Task B4 and interacting with backend hosted on AWS endpoint

5) FoodTracker.json: A Postman collection of API payload to test API calls easily

6) .github/workflows : demonstration of automation testing for Task B2

Step 2: Open up the terminal /console window and go to the directory in Step 1. Then, ensure you are inside the project folder of CS3219_OTOT_B before moving on by executing following command:

cd CS3219_OTOT_B

Similar output should be obtained:

```
(base) tkx@Tays-MacBook-Pro CS3219_OTOT_B %
```

Instructions on how to:

(i) Run the API locally, including Postman calls used to demonstrate a working API.

Step 1: Change into the backend folder directory where local CRUD implementation takes place in the NodeJS application by executing following command:

cd backend

Expected output:

```
(base) tkx@Tays-MacBook-Pro CS3219_OTOT_B % cd backend
(base) tkx@Tays-MacBook-Pro backend %
```

Step 2: Install the node application dependencies as specify in the package.json file required by backend NodeJS application using the following command:

npm install

Preview of package.json file:

```
13    "dependencies": {
14      "cors": "^2.8.5",
15      "dotenv": "^10.0.0",
16      "express": "^4.17.1",
17      "mongoose": "^6.0.5",
18      "nodemon": "^2.0.12"
19    },
20    "devDependencies": {
21      "chai": "^4.3.4",
22      "chai-http": "^4.3.0",
23      "mocha": "^9.1.1",
24      "supertest": "^6.1.6"
25    }
26  }
```

Step 3: Next, start up the backend application using the following command:

npm start

I have added the start script using the nodemon npm package which enables hot reloading where the NodeJS application can be restarted automatically whenever there is a file change. index.js is the entry point of the application.

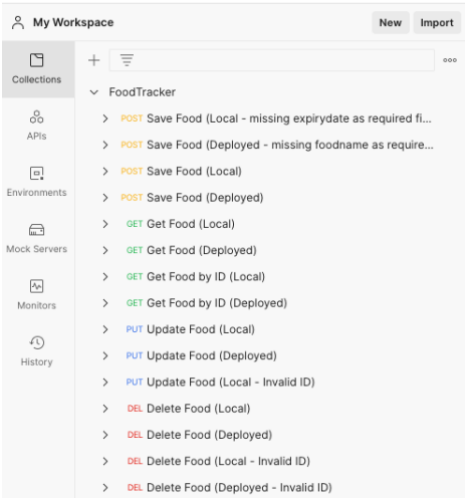
```
6     "scripts": {
7       "start": "nodemon index.js",
```

Expected output:

```
(base) tkx@Tays-MacBook-Pro backend % npm start

> backend@1.0.0 start
> nodemon index.js

[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Db connected successfully
Running FoodTracker backend on port 8080
```

Step 4: Open the Postman app and click Import then in the pop-up window select Upload File under the File tab to import the FoodTracker.json file.

You should see a screen like this:



Note:
Local means the backend application is served at local endpoint which is
http://localhost:8080/api/food
while the deployed means the backend has been deployed to AWS cloud endpoint which is
https://doduzz3kdg.execute-api.ap-southeast-1.amazonaws.com/dev/api/food

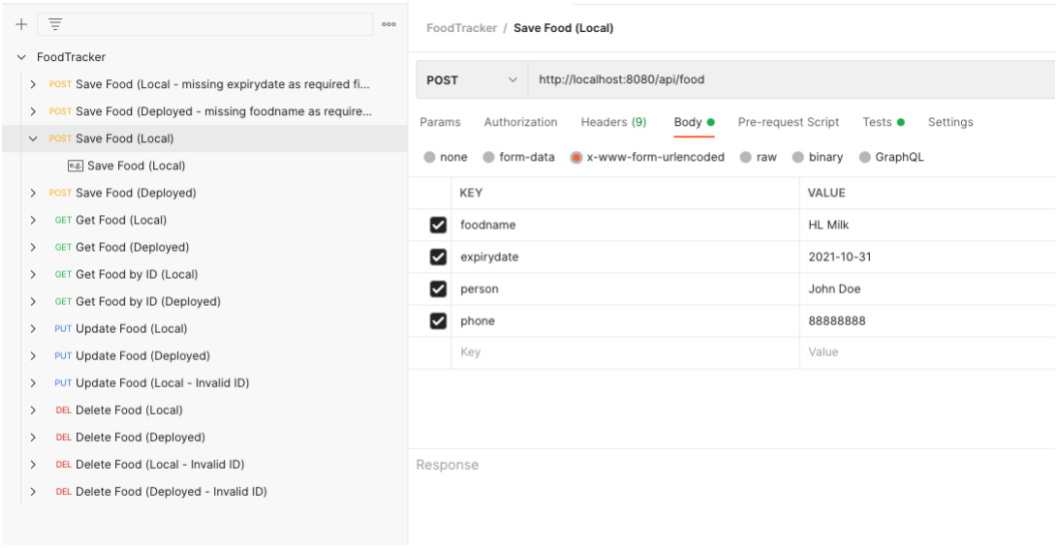
There is no need to do serverless deploy since the backend application has been deployed onto the cloud using AWS lambda and serverless which will be explained later.

Step 5:

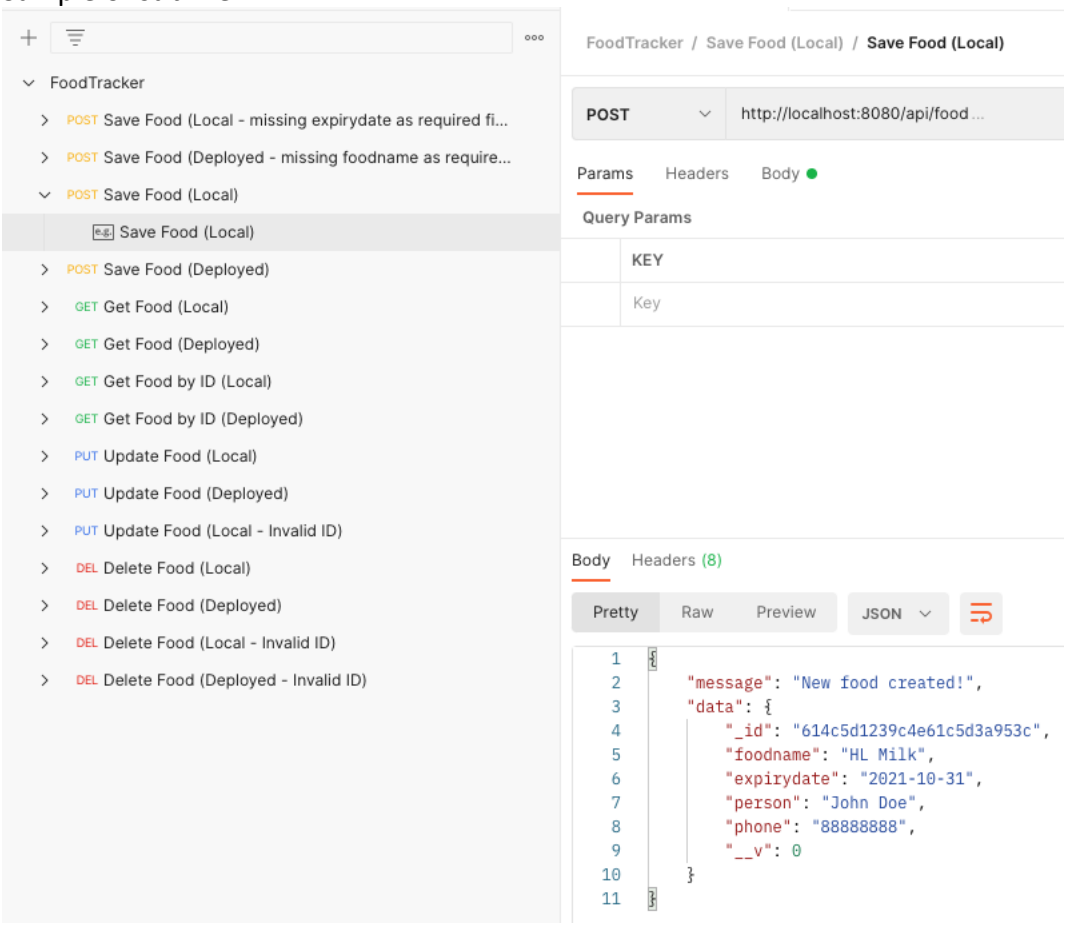
In order to try out the CRUD calls, you will go down each individual API call in order.

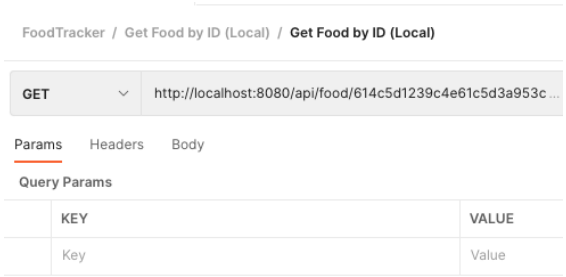
For each API call there is a main file where the key and value has been defined when the API call is sent and the sub-file is the response from the API call that I have executed.

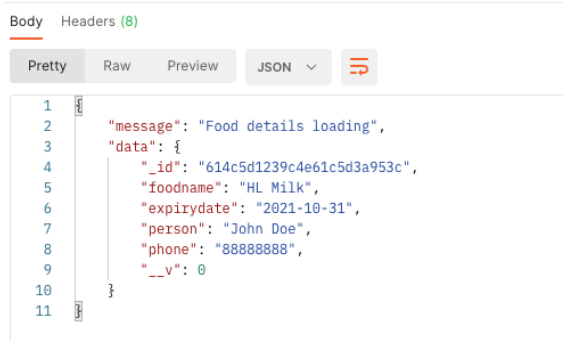
Sample of main file:



Sample of sub file:





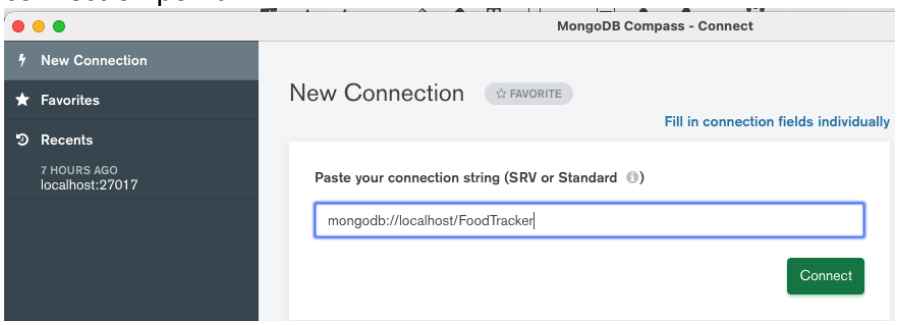


Note: For Get Food by ID, Update Food and Delete Food API calls, the _id which is generated randomly by MongoDB during runtime will be attached to the back of the API request URL. In the picture shown above, the _id is **614c5d1239c4e61c5d3a953c.**

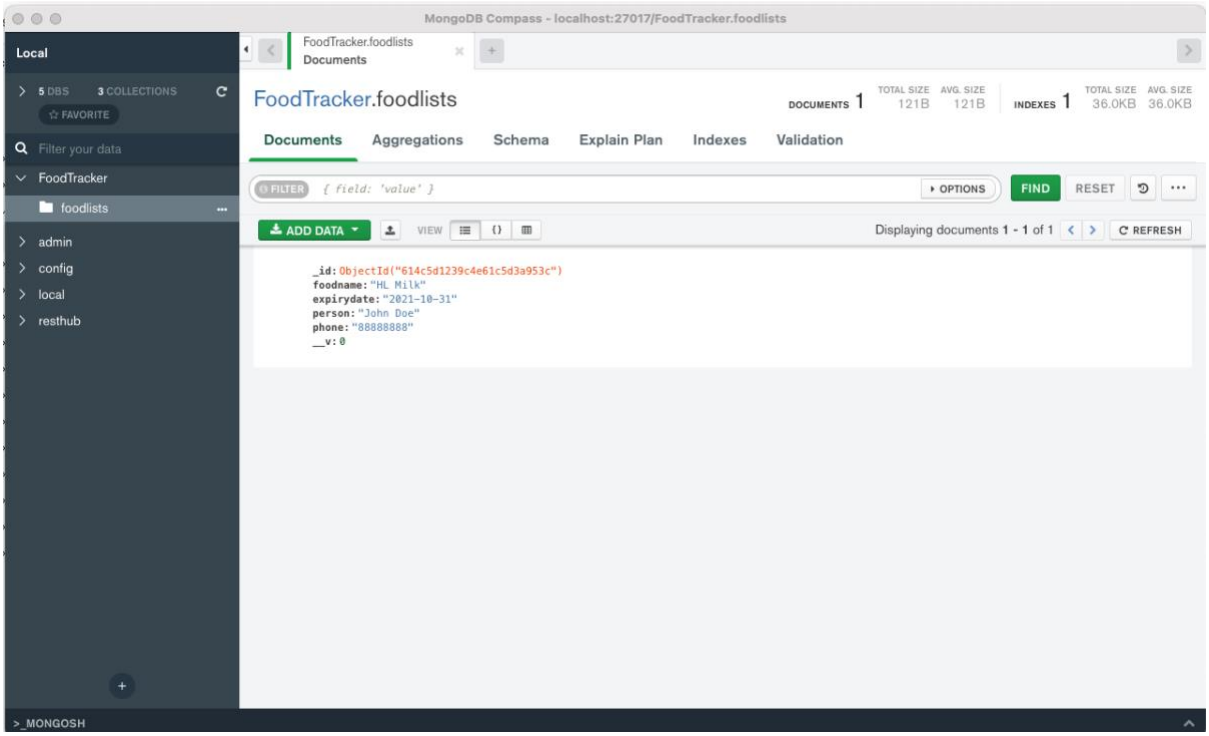
Hence, please do Get Food where the list of all food in the local or cloud databases can be displayed first, note down the targeted _id in a Notepad or Text Editor before carrying out the subsequent operations such as Food by ID, Update Food and Delete Food API calls and replaced the updated _id with my current _id at the back of the request URL before sending the API call.

Optional but you can use MongoDB Compass to view the localhost database data

For the local MongoDB connection, I used **mongodb://localhost/FoodTracker** as the connection point.



Click into FoodTracker collection and select the foodlists table where you will be able to see the data clearly.



ii. Access the deployed API

This is where I create my own Cluster in MongoDB Atlas which is a Database-as-a-Service

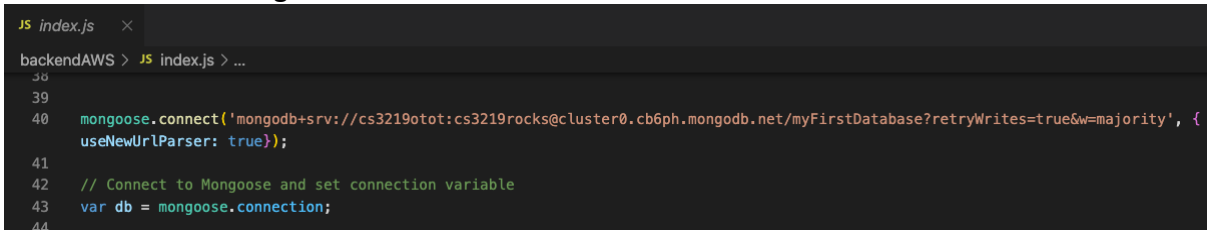
Step 1: Instructions to setup is as follows: https://docs.atlas.mongodb.com/getting-started/

Step 2: From the instructions guide of setting up the network, I choose **wildcard option** to allow IP address from everywhere

Step 3: After creating your own username and password for the cluster, open a separate terminal window and switched into the backendAWS folder.

Step 4: Do **npm install** to install the NodeJS application dependencies.

Step 5: After setting up the username and password from instructions guide within step 1, Open a code editor and navigate into index.js of the backendAWS folder and change the connection URL string in line 40.



Step 6: By now, if you have signed up for a AWS account, you can refer to this link https://stackify.com/aws-lambda-with-node-js-a-complete-getting-started-guide/ where from the steps 1 to 7, it will demonstrates how to set up IAM role, giving administrator access, setting up AWS CLI to configure access key and access ID.

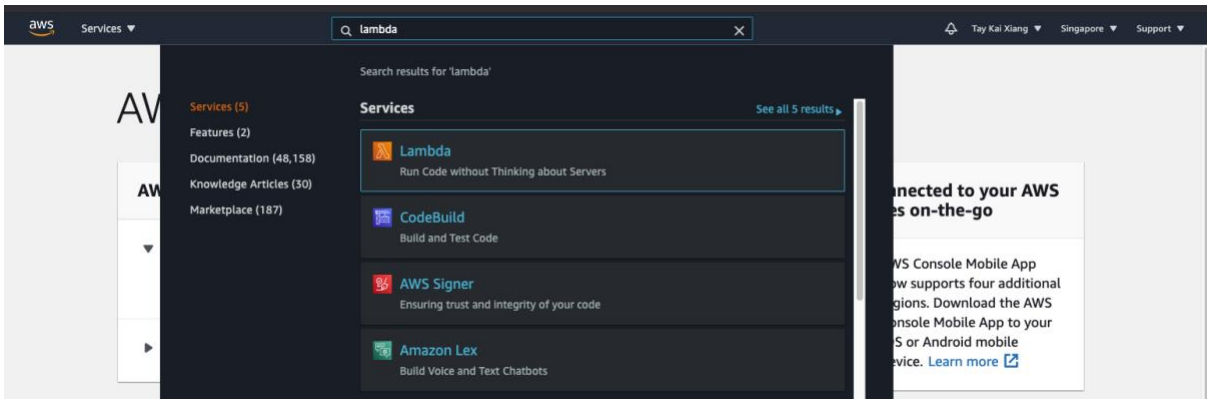
Step 7: Execute the command **serverless deploy** to deploy the backend onto AWS cloud again.

In the terminal output, you should see your own AWS cloud endpoint link under the endpoints in Service Information section.

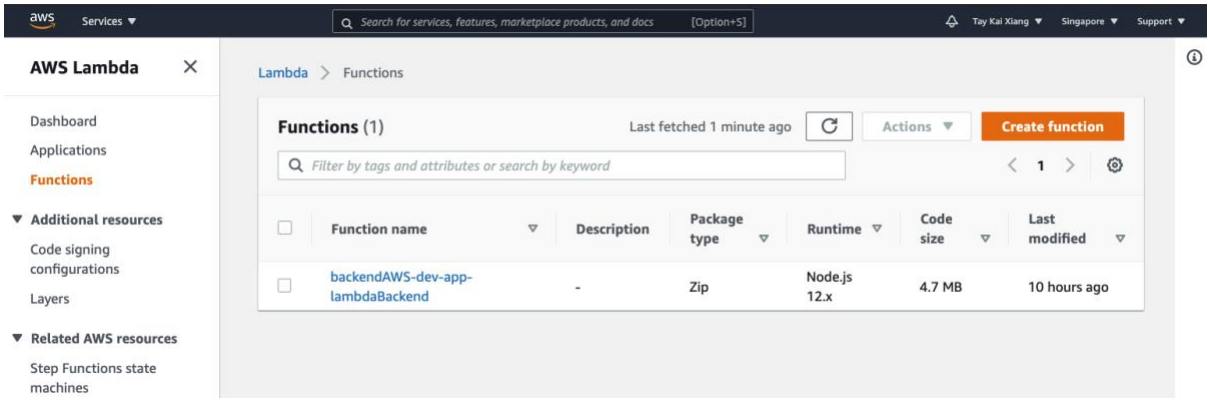
```
(base) tkx@Tays-MacBook-Pro backendAWS % serverless deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service backendAWS.zip file to S3 (4.96 MB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.............
Serverless: Stack update finished...
Service Information
service: backendAWS
stage: dev
region: ap-southeast-1
stack: backendAWS-dev
resources: 12
api keys:
  None
endpoints:
  ANY - https://doduzz3kdg.execute-api.ap-southeast-1.amazonaws.com/dev/{proxy+}
functions:
  app-lambdaBackend: backendAWS-dev-app-lambdaBackend
layers:
  None
```

This is my own endpoint link:
https://doduzz3kdg.execute-api.ap-southeast-1.amazonaws.com/dev/

Step 8: If you have signed up for your AWS free tier account, you can login and you should see the AWS Management Console. You can search for AWS Lambda in the search bar.

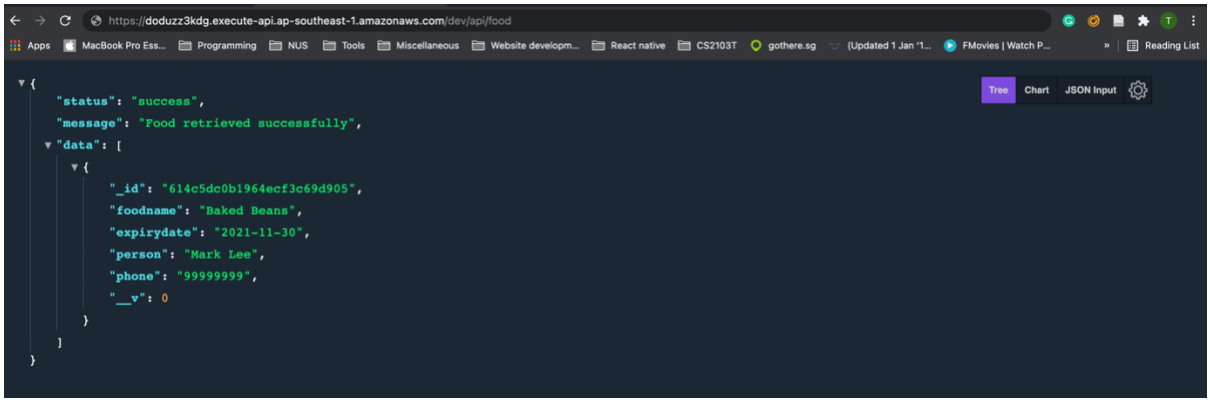


Step 9: Inside AWS Lambda, verify that the deployed Lambda function is present where the whole backend application has been packaged as a zip.



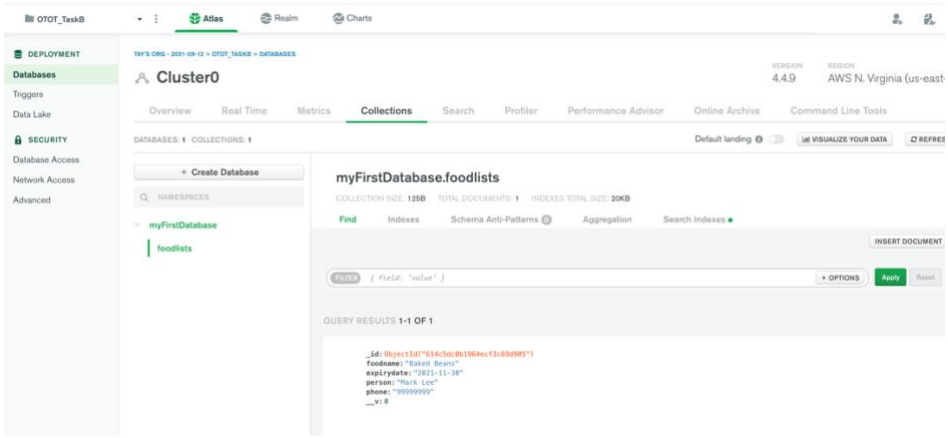
Step 10: Later, add **api/food** to the back of the link that you have obtained in Step 9. If everything is successful, you will be able to see this cloud endpoint. I have used JSON Viewer Pro from the Chrome Web Store to help me see the JSON data file in a better manner.

Here is the link to download the JSON Viewer Pro: https://chrome.google.com/webstore/detail/json-viewer-pro/eifflpmocdbdmepbjaopkkhbfmdgijcc



Similarly, if you have signed up and configure the MongoDB Atlas in Step 1 and 4, you can see the online database data in MongoDB Atlas.

This is a screenshot of my own cluster data.



This is the serverless.yml file which is required to configure the deployment of AWS lambda using the serverless framework. The lambda function is called **app-lambdaBackend**.

```
! serverless.yml ×
backendAWS > ! serverless.yml
1   service: backendAWS
2
3   provider:
4     name: aws
5     runtime: nodejs12.x
6     apiGateway:
7       minimumCompressionSize: 1024 # Enable gzip compression for responses > 1 KB
8     stage: dev
9     region: ap-southeast-1
10
11  functions:
12    app-lambdaBackend:
13      handler: handler.handle
14      events:
15        - http:
16            path: /{proxy+}
17            method: any
18            cors: true
19
```

I have added the method to be **any** within the events section in the lambda function so this lambda function can serve any http request actions inside the controller. We do not have to configure multiple AWS lambda functions to serve the CRUD operations.

I have also set the region to be ap-southeast-1 which is Singapore.

In the configuration, apart from using serverless, I have also used aws-serverless-express to package and create the Lambda function, then configure a simple proxy API using Amazon API Gateway and integrate it with the Lambda function.

Here are the references for serverless and aws-serverless-express:
https://www.serverless.com/framework/docs/providers/aws/guide/serverless.yml/
https://www.npmjs.com/package/aws-serverless-express

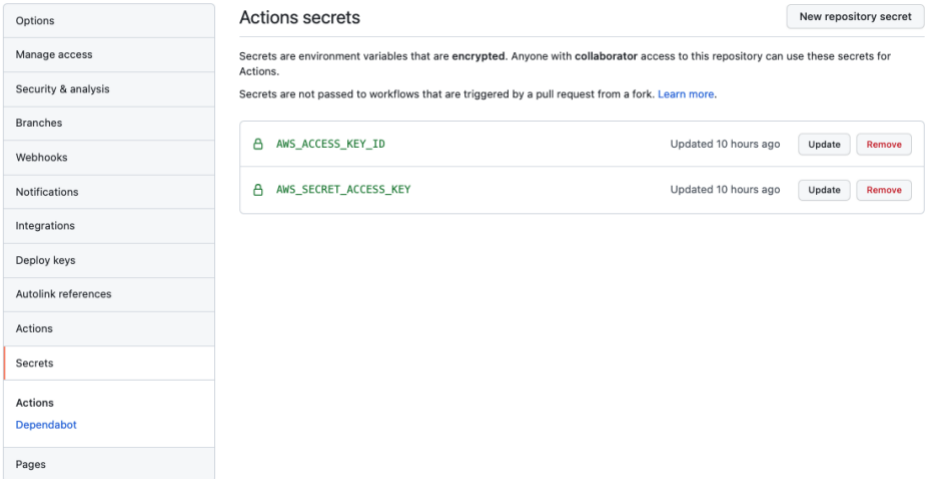
Here is the link to the CD for task B3 where the backendAWS application will run through the CI pipeline using GitHub Actions before being deployed onto the AWS.
https://github.com/kaixiangtay/CS3219_OTOT_B/blob/main/.github/workflows/CD.yml

49 lines (40 sloc) | 1.24 KB Raw Blame

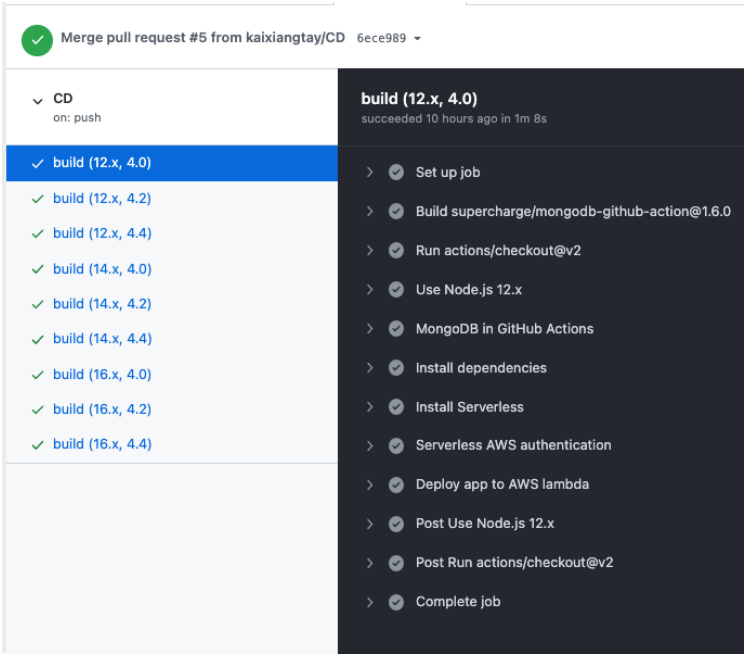
```yaml
name: CD

on:
  push:
  pull_request:
    branches:
      - backendAWS

jobs:
  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [12.x, 14.x, 16.x]
        mongodb-version: ['4.0', '4.2', '4.4']

    steps:
    - uses: actions/checkout@v2
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v2
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
        cache-dependency-path: backendAWS/package-lock.json
    - name: MongoDB in GitHub Actions
      uses: supercharge/mongodb-github-action@1.6.0
      with:
        mongodb-version: ${{ matrix.mongodb-version }}
        mongodb-port: 27017

    - name: Install dependencies
      working-directory: backendAWS
      run: npm ci
      env:
        CI: true

    - name: Install Serverless
      working-directory: backendAWS
      run: npm install -g serverless

    - name: Serverless AWS authentication
      working-directory: backendAWS
      run: serverless config credentials --provider aws --key ${{ secrets.AWS_ACCESS_KEY_ID }} --secret ${{ secrets.AWS_SECRET_ACCESS_KEY }}

    - name: Deploy app to AWS lambda
      working-directory: backendAWS
      run: serverless deploy function --function app-lambdaBackend
```

Do note: I have stored the AWS_Access_KEY_ID and AWS_SECRET_ACCESS_KEY using GitHub secrets. The AWS_Access_KEY_ID and AWS_SECRET_ACCESS_KEY is the same that has been configured in Step 5 above.

Place to store the GitHub Secrets under GitHub Profile section:



Here is the link of a sample of the pipeline check:
https://github.com/kaixiangtay/CS3219_OTOT_B/pull/6/checks



Note: It will only happen when there are new changes made to merge into the backendAWS branch or you can click re-run all jobs from the link above.

Under Install dependencies, I used **npm ci** not only it is faster but it ensures that my dependencies are a clean installation suitable for automation environment.

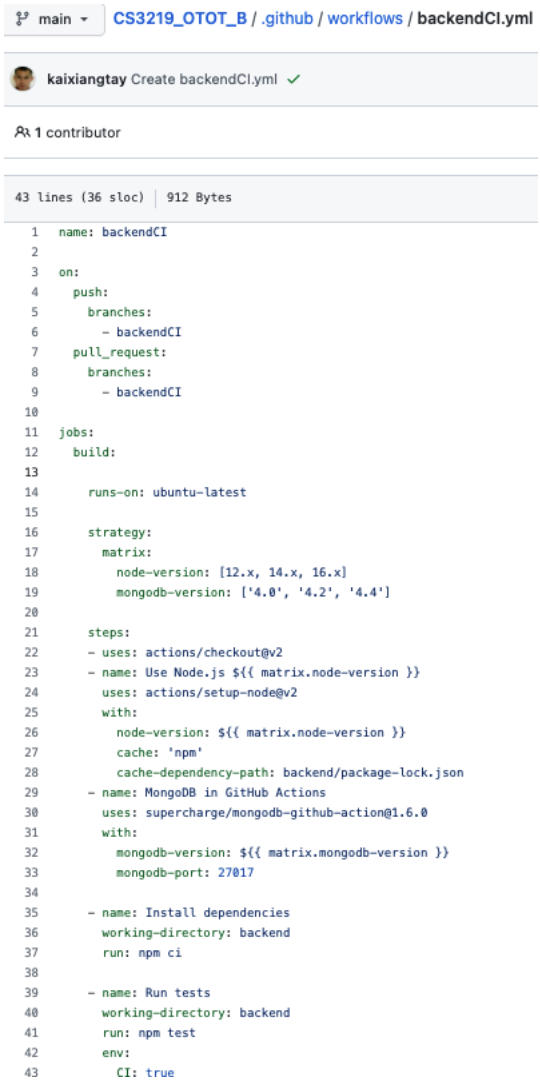
For the NodeJS and MongoDB configuration in the.yml file, I have used this as reference
https://github.com/marketplace/actions/mongodb-in-github-actions

There are 9 different checks to permutate between various versions of NodeJS and mongo-db. actions/setup-node@v2 is used to setup NodeJS environment in the pipeline, supercharge/mongodb-github-action@1.6.0 is used to setup MongoDB environment in the pipeline.

iii. Run tests locally and via Travis

Here is the link of the CI testing for Task B2 for backend application folder: https://github.com/kaixiangtay/CS3219_OTOT_B/blob/main/.github/workflows/backendCI.yml

Here is the screenshot of the backendCI.yml:



main ▾ CS3219_OTOT_B / .github / workflows / backendCI.yml

kaixiangtay Create backendCI.yml ✓

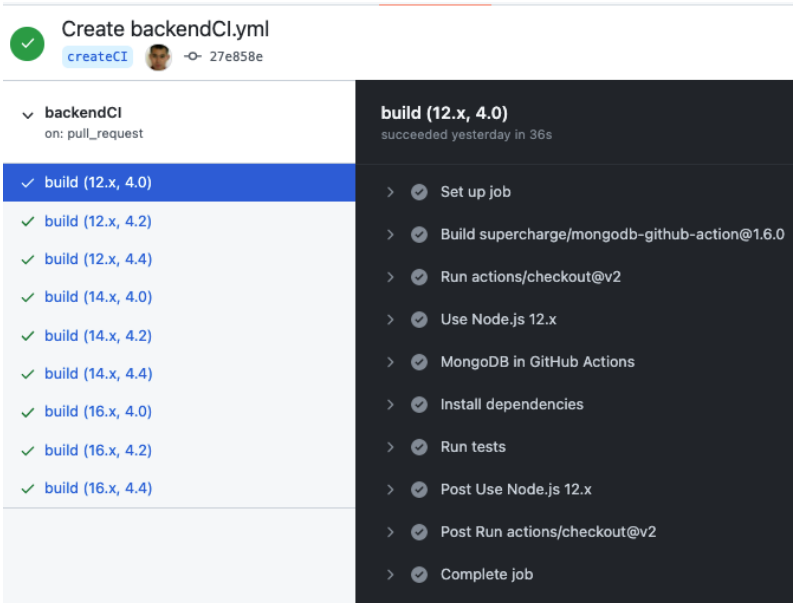
1 contributor

43 lines (36 sloc) | 912 Bytes

```yaml
name: backendCI

on:
  push:
    branches:
      - backendCI
  pull_request:
    branches:
      - backendCI

jobs:
  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [12.x, 14.x, 16.x]
        mongodb-version: ['4.0', '4.2', '4.4']

    steps:
    - uses: actions/checkout@v2
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v2
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
        cache-dependency-path: backend/package-lock.json
    - name: MongoDB in GitHub Actions
      uses: supercharge/mongodb-github-action@1.6.0
      with:
        mongodb-version: ${{ matrix.mongodb-version }}
        mongodb-port: 27017

    - name: Install dependencies
      working-directory: backend
      run: npm ci

    - name: Run tests
      working-directory: backend
      run: npm test
      env:
        CI: true
```

The critical explanations for the various configurations within the .yml has been mentioned above.

Here is the link for the CI pipeline:
https://github.com/kaixiangtay/CS3219_OTOT_B/runs/3672462622



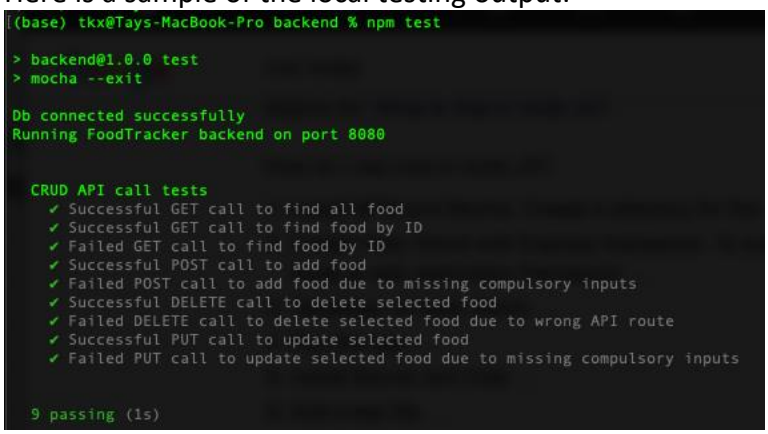
Note: It will only happen when there are new changes made to merge into the backendCI branch or you can click re-run all jobs from the link above.

For local testing, I did it using mocha, a JavaScript test framework running on Nodejs and chai for assertions of outputs as well as supertest for HTTP assertions.

Step 1: Ensure there is only a single terminal and the other terminal instances that run the backend folder previously has been closed using ctrl + c for Windows / command + c for Mac.

Step 2: Assuming you have installed the dependencies in (i) section for the backend folder, we can just execute **npm test** command to initiate local testing.

Here is a sample of the local testing output:



iv. Set up frontend

For task B4, I have used React JavaScript library for the frontend coupled with reactstrap for styling.

Here are the references:
https://reactjs.org/
https://reactstrap.github.io/

Option A: Connect frontend to local backend MongoDB service

Step 1: Open a terminal window, navigate to the CS3219_OTOT_B/backend directory.

Step 2: If you have done previous part (i). You can ignore this step, or else execute **npm install** command to install the project dependencies in the backend folder.

Step 3: Run **npm start** to start the local backend service.

Step 4: Open another terminal window, navigate to the CS3219_OTOT_B/frontend directory.

Step 5: Execute **npm install** command to install the project dependencies in the frontend folder.

Step 6: Run **npm start** to start the frontend application.

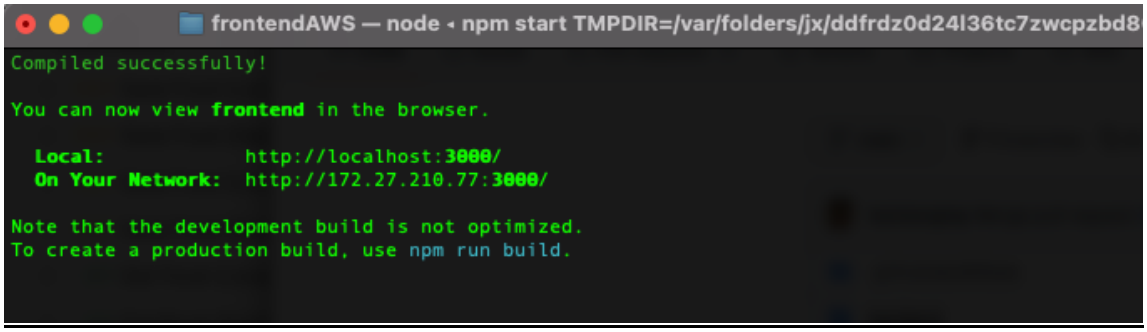
Option B: Connect frontend to AWS hosted backend MongoDB service

Pre-requisites: Complete (ii) setup

Step 1: Open another terminal window, navigate to the CS3219_OTOT_B/frontendAWS directory.

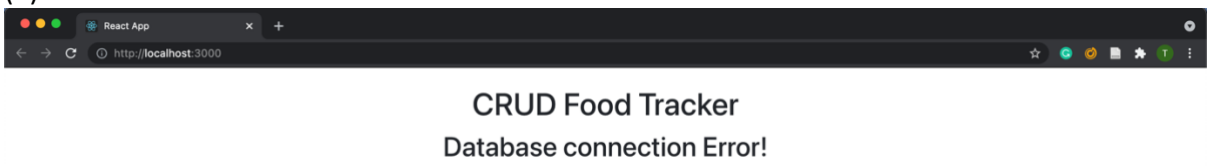
Step 2: Execute **npm install** command to install the project dependencies in the frontendAWS folder.

Step 3: Run **npm start** to start the frontendAWS application. Below is the screenshot of the output console you should see which is the default React endpoint **http://localhost:3000/**



Troubleshoot

(a) Database connection error



Option A

If you encountered this error, it means the local backend service is not connected to local MongoDB **mongodb://localhost/FoodTracker.**

Open another terminal window, navigate to the CS3219_OTOT_B/backend directory.

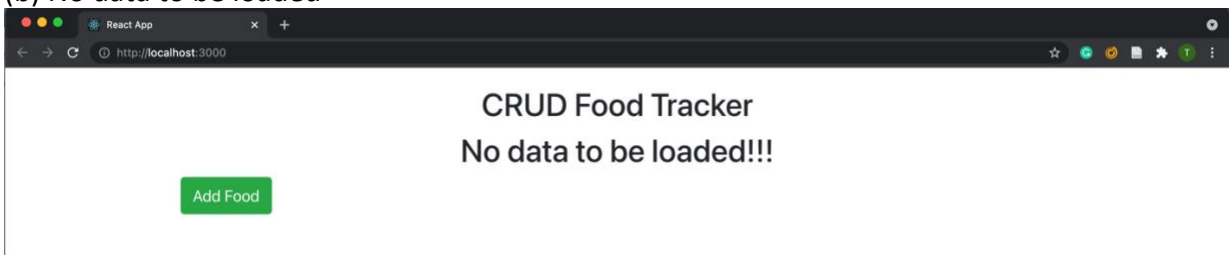
Then, execute **npm start** to start the backend service.

Option B

If you encountered this error, it means the AWS-hosted backend service is not connected to the cloud MongoDB Atlas.

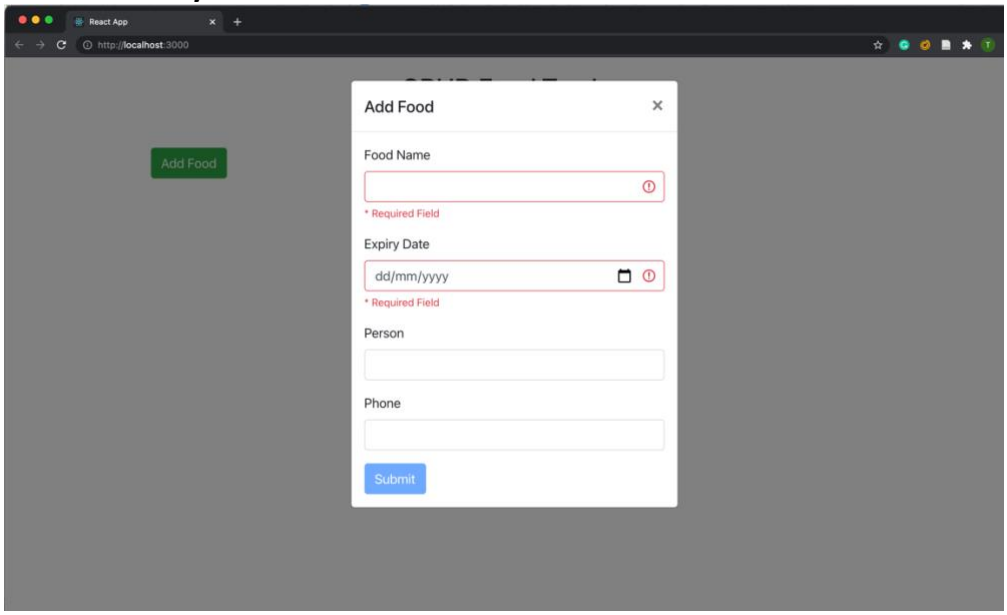
Please check the connection string URL in index.js of backendAWS folder, which is the entry point of the backend application.

(b) No data to be loaded

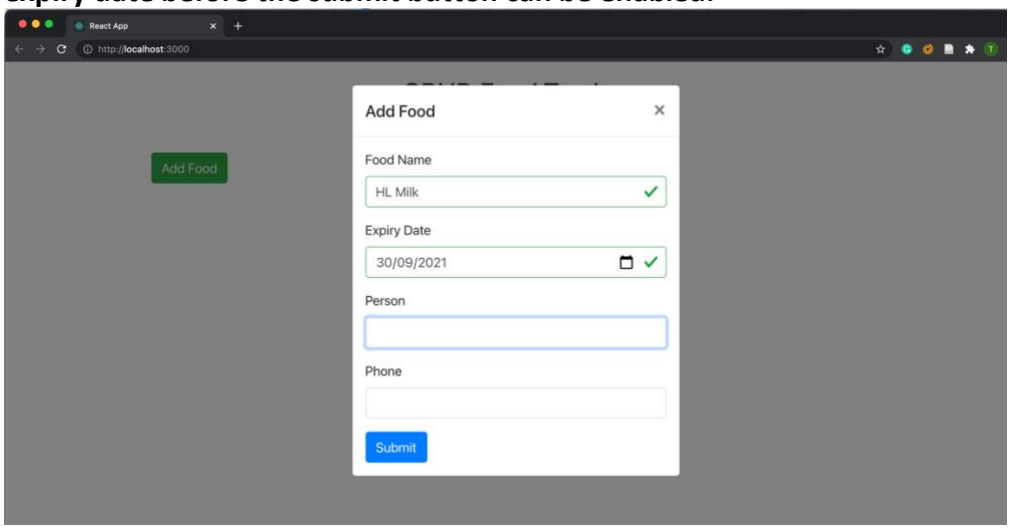


There is nothing wrong to be concerned. You can go ahead to add the food accordingly. This is just an indicator/feedback that the backend database is empty.

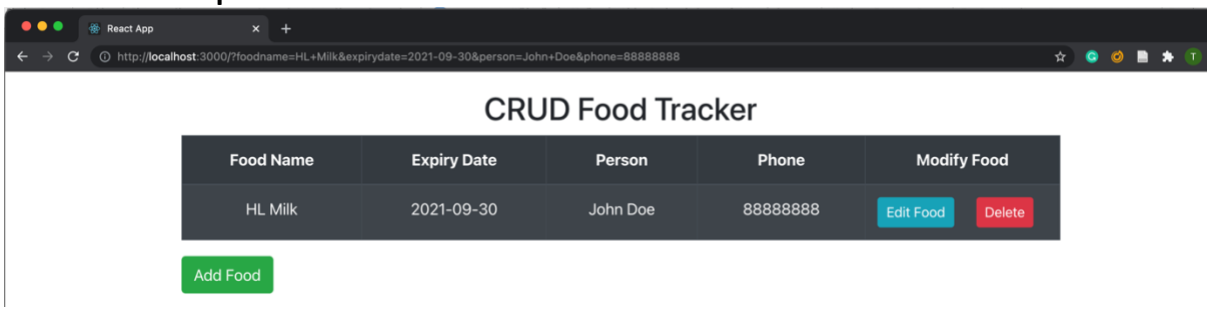
Preview after you click the add food button:



Note: Submit button is disabled by default. It is required to enter the food name and expiry date before the submit button can be enabled.



Here is an example of a successful submission:

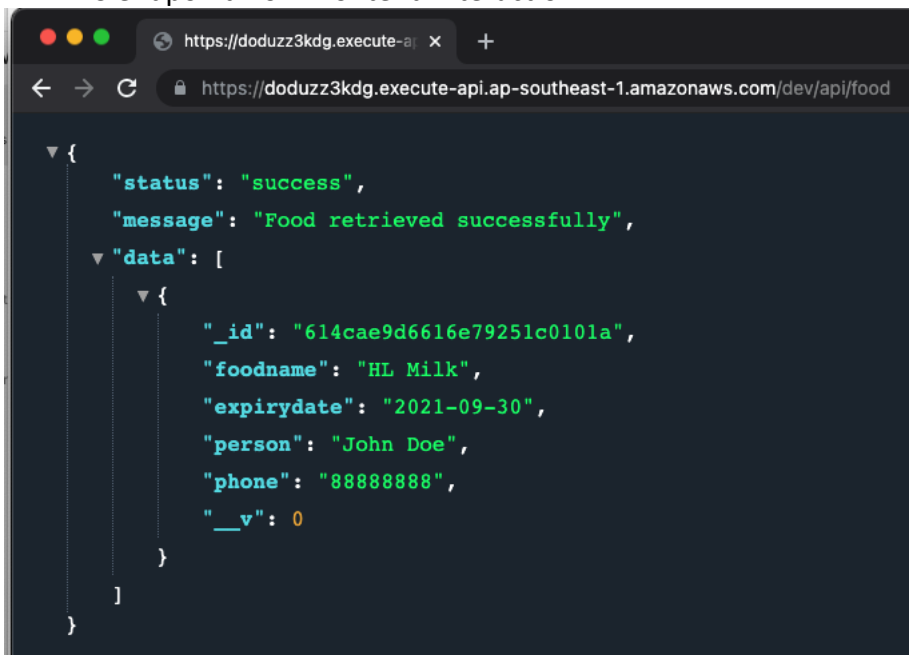


Notice for each entry, the edit food and delete food button will be available.

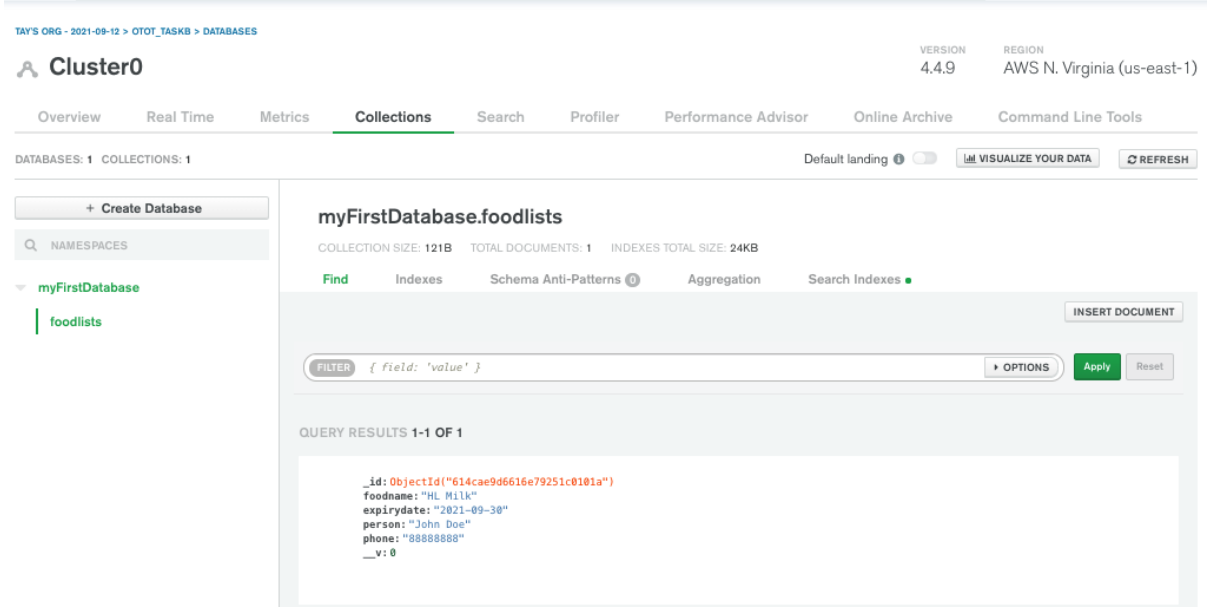
For option A, you can use the MongoDB Compass to verify the data.

For option B, Here you can open the AWS hosted endpoint or MongoDB Atlas to verify the entry data.

An AWS endpoint from frontend interaction:

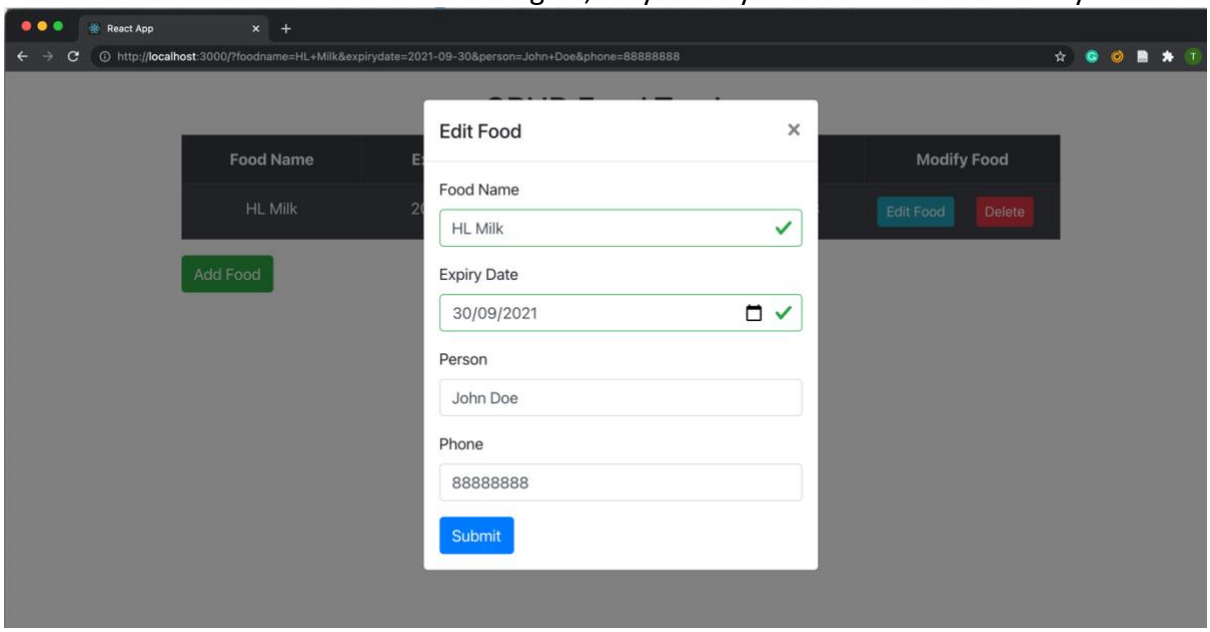


An entry in a database on MongoDB Atlas from frontend interaction:



Bonus:

Once, I clicked on the Edit Food for the same entry. The existing data will be available so the user will not need to fill in all the fields again, only modify certain fields in the entry.



Lastly, you can click the delete to delete the food entry when you no longer need it anymore. This will fulfil the CRUD Food Tracker application.