# REST API to make CRUD operations on restaurants information

I used ***Node.js*** (version v11.11.0 with ***npm*** version 6.7.0), ***Express web framework*** (version 4.16.4), ***Mocha*** (version 6.0.2) to implement the application.

By navigating to the "crude_web_app" folder through terminal, we can run:

- "*npm start*" to start the application or,
- "*npm test*" to run the tests for the application.

After running "npm start", we can perform the included CRUD operations as below.

## Included CRUD operations

We can use cURL, POSTMAN or any similar tool to interact with the API's. For e.g. let's see how cURL can be used from terminal as specified below for each API. Returned responses are in JSON format.

- An endpoint that lets the client get a list of all restaurant names.
  ***curl -X GET http://localhost:3000/display***
- An endpoint that lets the client get more info on a single restaurant based on restaurant *'id'*. All attributes of that restaurant will be displayed.
  ***curl -X GET http://localhost:3000/display/<id>***
- An endpoint that accepts a POST request to add new restaurants. We can put a single restaurant detail as an object on JSON file by giving it a filename and we can execute it as follows by running the *cURL* from the filename's directory:
  ***curl -X POST -H "Content-Type:application/json" http://localhost:3000/write/addnew –d @filename***
- Functionality to delete restaurants through the API by using restaurant *'id'*.
  ***curl -X DELETE http://localhost:3000/write/<id>***
- Functionality to fetch a sorted list of restaurants based on restaurant '*rating'* and '*name'*. Restaurants will be sorted based on their rating from highest to lowest. Restaurants that have same rating are sorted based on their names.
  ***curl -X GET http://localhost:3000/display/sorted***
- Functionality to fetch a filtered list of restaurants based on restaurant '*rating'*. Only restaurants whose ratings are above 4.2 will be displayed.
  ***curl -X GET http://localhost:3000/display/filtered***

## Short explanation of my design decisions

I have an Express application instance in the *'src/index.js'* file. To modularize the implementation, I put the database schema in *'/src/models/mongo.js'* file and exported it to my Express application instance. The Resources/API's that are used to display all restaurants, single restaurants, sorted and filtered restaurants are only reading from database and making no changes. So they are placed in a same URI called '*display*' *(http://localhost:3000/display).* In contrast with this, Resources/API's for adding new restaurant and deleting restaurant makes change to the existing database. Hence, they are placed in a URI called 'write' *(http://localhost:3000/write).* Inside *'src'* folder, I have *'routes'* folder which will be used to make modular routes to the resources with Express Router. This folder contains an entry point file (*routes/index.js*) and two additional files that are used for modularized implementation of Resources to read (*routes/display.js*) from database and make changes (*routes/write.js*) on the database: this files will be exported as objects

through the entry point file to be used by the Express application instance. This two (write and display) files in '*routes'* folder also have their own dedicated URI, so this URI is used to mount them to the Express application instance (*/src/index.js*). As a summery, the 'display' URI path contains all the GET resources, whereas the 'write' URI path contains POST and DELETE resources.

For testing, the application uses different database (viaplay-test) other than the original database (viaplay) so that it will not use our real data and make changes to them. The '*src/_config.js*' holds the two names of database addresses that are going to be used for development and for test. The *'.env'* file is used to store sensitive information used on applications, in this application it only contains the name of the port the application uses to listen.

I made the *'id'* field in the database schema to be unique and required. Because throughout the application, I used *'id'* to uniquely identify each element (row) in the database. *Package.json* file contains the scripts that will be used to start and test the application, and contains dependency information and tools version I used.

## About the tests

As mentioned above, a new database named *'viaplay-test'*, will be created every time we run the test which will be dropped after end of the test. This means the *'viaplay'* database will not be accessed during the testing period. I created two testing files in the *'test'* folder. *'test/write.test.js'* file is used to test the resources on *'src/routes/write.js'* where as *'test/display.test.js'* file is used to test the resources provided by *'src/routes/display.js'*. I created 9 tests: 5 of them test the GET resources, 2 of them test POST resource and the rest 2 tests test the DELETE resource. Each test is BDD style (using *mocha* and *chai*) focusing on how the methods behave on request. Before each test case runs, a database collection that contain test data is created and dropped immediately after it ended so that a new test data will be provided for the next test case. In addition to this common test data, I also tried to embed another test data inside each test case's in order to have at least one additional record. Doing so is more important in testing the SORT and FILTER resources, which need more than one record to have a meaning.

This assignment was very interesting and I thank everyone who is involved in this recruitment process. I plan to add more features to the current status. For e.g. making it more efficient algorithm wise, adding more test varieties (mock tests) and other features which I believe would enhance my knowledge and skill on web applications, web services and Node.js. Though it is not completely working due to shortage of time, below is my progress of writing the Dockerfile to dockerize the application.

```
FROM node:8
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run test
RUN npm run start
EXPOSE 27017
CMD [ "npm", "start" ]
```

***Musie Ebuy***