# Lab 1

**Question 1: Open the following URL’s within the web browser and take screenshots; place them inside the DOCX file created.**

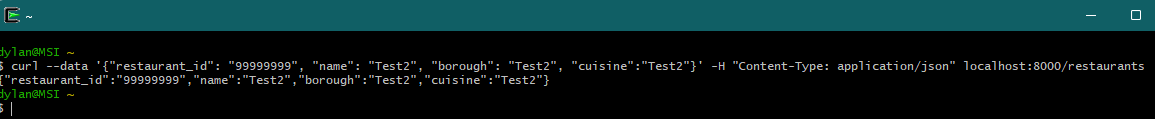
* http://localhost:8000/restaurants/40356018
* <http://localhost:8000/restaurants/99999999>

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

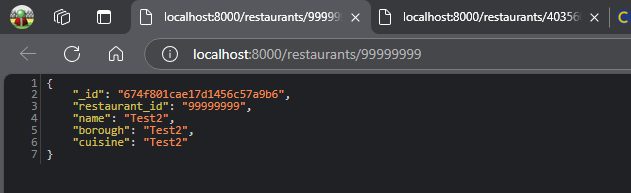
Description automatically generated

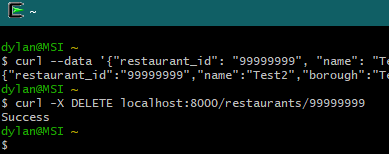
**Question 2: Run the following command on your command line tool; take a screenshot of the output and place it inside the DOCX file created; ensure the whole command be executed in one (1) line.  
curl --data '{"restaurant\_id": "99999999", "name": "Test2", "borough": "Test2", "cuisine":  
"Test2"}' -H "Content-Type: application/json" localhost:8000/restaurants**



**Question 3: Open the following URL within the web browser and take screenshot; place it inside the DOCX file created. You will see  
a new restaurant created.**

* <http://localhost:8000/restaurants/99999999>



**Question 4: Run the following command on your command line tool; take a screenshot of the output and place it inside the DOCX file created; ensure the whole command be executed in one (1) line.  
curl -X DELETE localhost:8000/restaurants/99999999** 

**Question 5: Open the following URL within the web browser and take screenshot; place it inside the DOCX file created. You will see the restaurant you created be now deleted.**

* [**http://localhost:8000/restaurants/99999999**](http://localhost:8000/restaurants/99999999)

A screenshot of a computer

Description automatically generated

**Question 6: Copy and paste the full server.js contents to the DOCX file created.**

import express from "express";

import cors from "cors";

import db from "./db/connection.js";

const PORT = process.env.PORT || 8000;

const app = express();

app.use(cors());

app.use(express.json());

app.get('/restaurants/:id', function (req, res) {

  const restaurant\_id = req.params["id"];

  db.collection("restaurants").findOne({

    restaurant\_id: restaurant\_id,

  })

    .then(value => {

      res.send(value)

    })

    .catch(() => res.status(500).send("Not Found"));

});

app.post('/restaurants', function (req, res) {

  const restaurant\_id = req.body['restaurant\_id'];

  const name = req.body['name'];

  const borough = req.body['borough'];

  const cuisine = req.body['cuisine'];

  db.collection("restaurants").insertOne({

    restaurant\_id: restaurant\_id,

    name: name,

    borough: borough,

    cuisine: cuisine

  }).then(result => result.acknowledged ?

    res.send({ restaurant\_id, name, borough, cuisine }) :

    res.status(500).send("Failed")

  ).catch(() => res.status(500).send("Failed"));

});

app.get('/restaurants', async (req, res) => {

  const borough = req.query['borough'];

  const cuisine = req.query['cuisine'];

  if (!borough || !cuisine) {

    res.status(422);

    res.send("Insufficient input.");

    return;

  }

  try {

    const restaurants = await db.collection("restaurants").find({

      borough: { $regex: new RegExp(`^${borough}$`, 'i') },

      cuisine: { $regex: new RegExp(`^${cuisine}$`, 'i') }

    }).toArray();

    if (restaurants.length === 0) {

      res.status(404);

      res.send("Error: No restaurants found");

      return;

    }

    res.status(200);

    res.json(restaurants);

    return;

  } catch (error) {

    res.status(500).send("Error: Unable to retrieve restaurants.");

  }

})

app.delete('/restaurants/:id', function (req, res) {

  const restaurant\_id = req.params["id"];

  db.collection("restaurants").deleteOne({

    restaurant\_id: restaurant\_id,

  }).then(result => result.acknowledged && result.deletedCount >= 1 ?

    res.send("Success") :

    res.status(500).send("Failed")

  ).catch(() => res.status(500).send("Not Found"));

});

app.listen(PORT, () => {

  console.log(`Server listening on port ${PORT}`);

});

**Question 7: Open the following URL’s within the web browser and take the screenshot; place it inside the DOCX file created**

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated A screenshot of a computer

Description automatically generated

**Question 8: Take a screenshot of the webpage on the web browser, and paste it inside the DOCX file created.** A computer screen shot of a computer program

Description automatically generated

**Question 9: Fill in the inputs with the following and click ‘Restaurant’ button at the top right corner. Take a screenshot of the result from the web browser and paste it inside the DOCX file created.**

A computer screen shot of a program

Description automatically generated

**Question 10: Fill in the inputs with the following and click ‘Mutation’ button at the top right corner. Take a screenshot of the result from the web browser and paste it inside the DOCX file created.** A screenshot of a computer

Description automatically generated

**Question 11: Modify the ‘Operations’ and ‘Variables’ from Question 8 to query the new restaurant whose id is ‘99999999\_1’ created in Question 9; run the query and take a screenshot from the web browser. Paste it inside the DOCX file created.**

A screenshot of a computer

Description automatically generated

**Question 12: Fill in the inputs with the following and click ‘Mutation’ button at the top right corner. Take a screenshot of the result from the web browser and paste it inside the DOCX file created.**

A computer screen shot of a computer program

Description automatically generated

**Question 13: Copy and paste the full schema.graphql contents to the DOCX file created.**

type Query {

  restaurant(restaurant\_id: String): Restaurant

  restaurants(borough: String, cuisine: String): [Restaurant]

}

type Mutation {

  createRestaurant(

    restaurant\_id: String

    name: String

    borough: String

    cuisine: String

  ): Restaurant

  deleteRestaurant(restaurant\_id: String): Boolean

}

type Restaurant {

  restaurant\_id: String

  name: String

  borough: String

  cuisine: String

}

**Question 14: Copy and paste the full resolver.js contents to the DOCX file created.**

**import db from "./db/connection.js";**

const resolvers = {

  Query: {

    async restaurant(\_, { restaurant\_id }) {

      let collection = await db.collection("restaurants");

      let query = { restaurant\_id: restaurant\_id };

      return await collection.findOne(query);

    },

    async restaurants(\_, { borough, cuisine }) {

      let collection = await db.collection("restaurants");

      let query = {};

      if (borough) query.borough = { $regex: `^${borough}$`, $options: "i" };

      if (cuisine) query.cuisine = { $regex: `^${cuisine}$`, $options: "i" };

      return await collection.find(query).toArray();

    },

  },

  Mutation: {

    async createRestaurant(\_, { restaurant\_id, name, borough, cuisine }, context) {

      let collection = await db.collection("restaurants");

      const insert = await collection.insertOne({

        restaurant\_id: restaurant\_id, name: name, borough: borough, cuisine: cuisine

      });

      if (insert.acknowledged) return { restaurant\_id, name, borough, cuisine };

      return null;

    },

    async deleteRestaurant(\_, { restaurant\_id }, context) {

      let collection = await db.collection("restaurants");

      const dbDelete = await collection.deleteOne({

        restaurant\_id: restaurant\_id

      });

      return dbDelete.acknowledged && dbDelete.deletedCount == 1 ? true : false;

    },

  },

};

export default resolvers;

**Question 15: Using the website (Apollo Client) in localhost:8000/graphql, query all Korean restaurants in Manhattan borough. Take a screenshot of the screen and place it inside the DOCX file created.**

A screenshot of a computer

Description automatically generated