

## CMPE 273- Lab 3- Yelp Recreation

**Name:** Bhavana Bangalore Sathyaprakash

**Id:** 014597245

**GitHub Code Link:** <https://github.com/BhavanaBS/CMPE273-Lab2>

### 1 Goals and purpose of the system

The goals the system includes the development and functional re-creation of Yelp. Yelp allows customers and restaurants to place and receive food orders.

Personas in Yelp:

- 1 Customer
- 2 Restaurant

A customer can create his profile with details such as date of birth, address, etc. It also allows him to search dishes available at various restaurants based on the filters of location, food deliver option and order them if he chooses to. Previous orders and order statuses are some of the other functionalities available to him.

A restaurant can create its profile with details of location, timings etc. It can also add the details of various dishes available and accept or reject orders it receives. A complete list of orders and events for a restaurant are also maintained.

### 2 System Design

The system is built using the below mentioned technologies:

#### 1 Frontend:

The front-end of the system is built using ReactJS to help in interaction between various components of the application. React Apollo connects front end to the GraphQL server which intern connects to the backend.

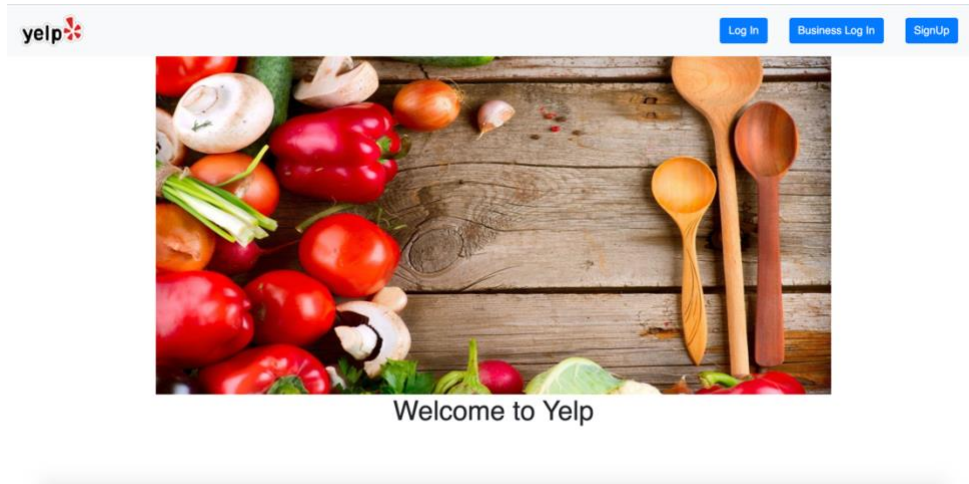
#### 2 Backend:

The back-end part of the application is built using NodeJS. It helps connect the frontend API to the NoSQL database. ESLint framework provides the static code analysis.

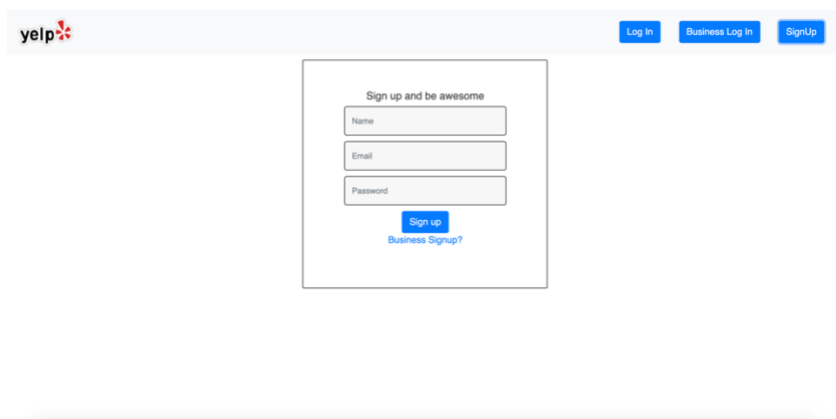
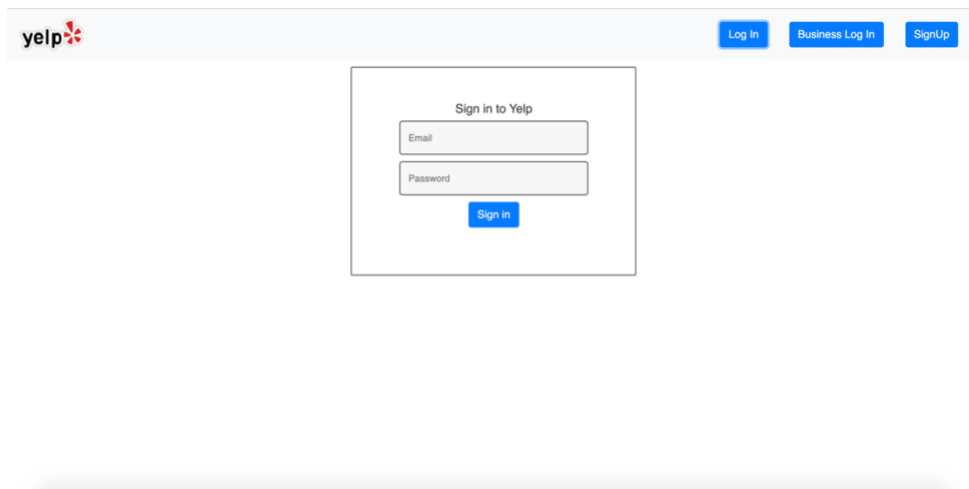
#### 3 Database:

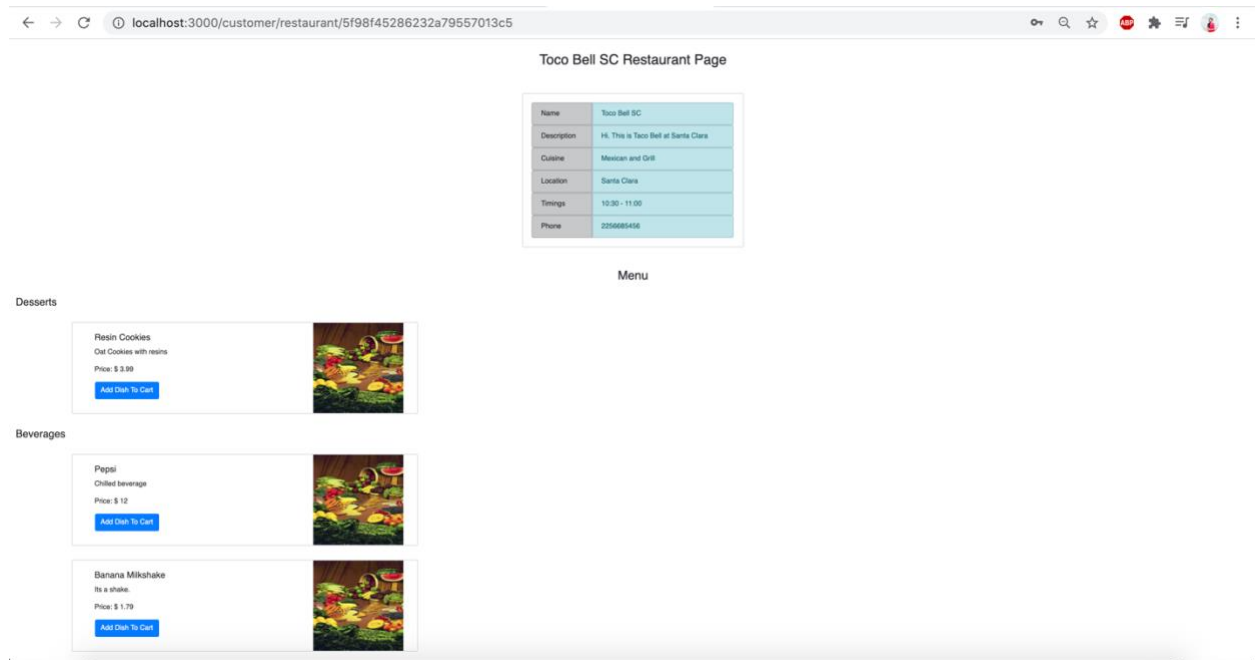
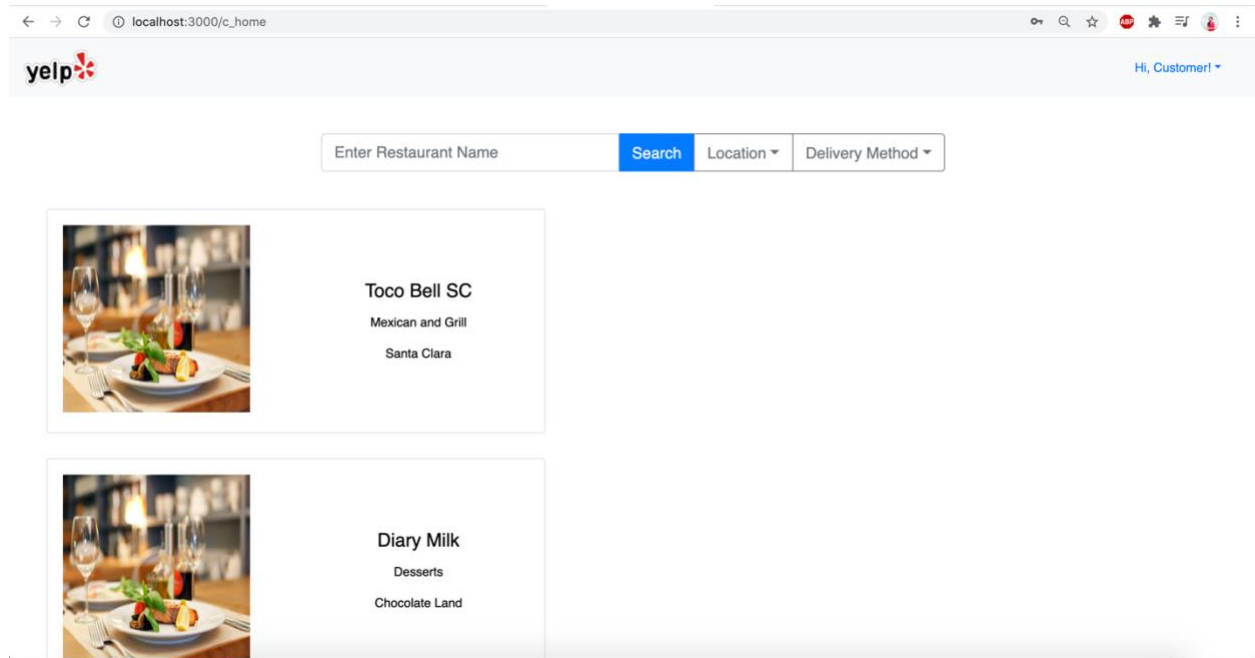
The database for the application is a NoSQL database MongoDB. This helps write simple function for database operations.

### 3 Screen from application



## Customer Pages:





Review Restaurant

Rating

Rating

Review

Please write a Review

Submit Review

Home

localhost:3000/c\_profile

Update Bhavana BS's Profile

Customer Name

Bhavana BS

About Me

I am Bhavana

City

Santa Clara

State

CA

Country

USA

Email

bhavana@gmail.com

Phone Number

2256685456

Date Of Birth

mm/dd/yyyy

Preferred Name

Bhavana

Join Date

Sat Dec 10

Blog url

https://kafka.apache.org/downloads

Favourite Hobby

Finding good tasty food

Favourite Restaurant

Taco Bell

Cancel

Update Details

yelp

Hi, Customer!

No dishes in cart!

Home

Buritto Updated


A super spicy buritto

Quantity: 1

Close

Add to Cart

←
→
↺
localhost:3000/c\_cart
🔍
☆
🔴
⚙️
☰
👤
⋮



Hi, Customer! ▾

Dishes in Cart

#	Dish Name	Dish Quantity	Price
1	Pepsi	1	12

Select Delivery Type ▾
Place Order
Home


←
→
↺
localhost:3000/c\_orders
🔍
☆
🔴
⚙️
☰
👤
⋮


Hi, Customer! ▾

Your Order History

Restaurant Name	Order Time	<div>Status</div> <div>Enter Status...</div>	Delivery Method	Dish Name	Quantity
Taco Bell SC	Dec 7, 2020	Order Received	Home Delivery	Banana Milkshake	2
Taco Bell SC	Dec 7, 2020	On The Way	Home Delivery	Banana Milkshake	2
Toco Bell SC	Dec 7, 2020	New Order	Home Delivery	Resin Cookies	1
Taco Bell SC	Dec 7, 2020	New Order	Home Delivery	Banana Milkshake	2

Home


⚙️
☰
👤
⋮

Hi, Customer! ▾

Home
Profile
Cart
Order History
Logout

Business

[Log In](#)[Business Log In](#)[Sign Up](#)

#### Sign in to Yelp as Business

[Sign in](#)[Log In](#)[Business Log In](#)[Sign Up](#)

#### Sign up and get rolling

[Sign up](#)

[Hi Restaurant!](#) ▼

[Home](#)[Profile](#)[Menu](#)[Orders](#)[Reviews](#)[Logout](#)

Toco Bell SC


Email Id	tacobell@sactaclara.com
Location	Santa Clara
Phone Number	2256685456
Description	Hi. This is Taco Bell at Santa Clara
Timings	10:30 - 11:00
Delivery Method	Home Delivery, Pick Up, Dine In

Edit Profile

#### Update Toco Bell SC's Profile

Restaurant Name	<input type="text" value="Toco Bell SC"/>
Description	<input type="text" value="Hi. This is Taco Bell at Santa Clara"/>
Location	<input type="text" value="Santa Clara"/>
Email	<input type="text" value="tacobell@sactaclara.com"/>
Phone Number	<input type="text" value="2256685456"/>
Timings	<input type="text" value="2256685456"/>
Delivery Method	<input type="text" value="Home Delivery"/>


Cancel Update Details



[Restaurant Menu](#)  
[Add Dish](#)

## Menu

### Desserts




*Resin Cookies*

Oat Cookies with resins

Price: \$ 3.99

[Edit](#)

### Beverages




*Pepsi*

Chilled beverage


Price: \$ 12

[Edit](#)



*Banana Milkshake*

It's a shake



[Restaurant Menu](#)  
[Add Dish](#)

## Add A Dish

Name:

Enter Dish Name

Description:

Description of the dish

Ingredients:

Ingredients in the dish

Price:

Enter Dish Price

Category:

Appetizer

[Cancel](#)
[Add Item](#)

[←](#)
[→](#)
[↺](#)

localhost:3000/r\_orders

🔍

☆


🔴

⚙️

📄

👤

⋮


[Hi Restaurant!](#)

## Your Orders

Customer	Order Time	Status	Update Status	Dish Name	Quantity	Delivery Method
		Enter Status...				
<a href="#">Customer Details</a>	Dec 7, 2020	Order Received	Order Recei ▾	Banana Milkshake	2	Home Delivery
<a href="#">Customer Details</a>	Dec 7, 2020	On The Way	On The Way ▾	Banana Milkshake	2	Home Delivery
<a href="#">Customer Details</a>	Dec 7, 2020	New Order	Order Recei ▾	Resin Cookies	1	Home Delivery
<a href="#">Customer Details</a>	Dec 7, 2020	New Order	Order Recei ▾	Banana Milkshake	2	Home Delivery

[Home](#)



- The screenshot shows the GraphQL IDE interface. On the left, the query editor contains a mutation query:

```
1 mutation{
2   addDish(restaurant_id:"5f98f45286232a79557013c5", name:"Banana Milkshake",
3     ingredients:"Banana, Milk", price:"1.79",
4     category:"Beverages", description:"Its a shake."){
5     message
6     status
7   }
8 }
9
```

On the right, the response editor shows the JSON response:

```
{
  "data": {
    "updateOrder": {
      "message": "ORDER_UPDATED",
      "status": "200"
    }
  }
}
```

The right sidebar shows the 'RootQueryType' schema with the following fields:

  - customer(customer\_id: String): Customer
  - restaurant(restaurant\_id: String): Restaurant
  - restaurants(input: String): [Restaurant]
  - menu(restaurant\_id: String): [RestDish]
  - dish(restaurant\_id: String, dish\_id: String): [RestDish]
  - reviews(restaurant\_id: String): [Review]
  - customerOrders(customer\_id: String): [Order]
  - restaurantOrders(restaurant\_id: String): [Order]

ke"%2C%...



< Schema

Mutation



No Description

FIELDS

```
addCustomer(  
  name: String  
  email_id: String  
  password: String  
): Status  
  
addRestaurant(  
  name: String  
  email_id: String  
  password: String  
  location: String  
): Status  
  
customerLogin(email_id: String, password: String): Status  
  
restaurantLogin(email_id: String, password: String): Status  
  
customerUpdate(  
  name: String  
  phone: String  
  dob: String  
  city: String  
  state: String  
  country: String  
  nick_name: String  
  about: String  
  favourite_restaurant: String  
  favourite_hobby: String  
  blog_url: String  
  email_id: String  
): Status  
  
restaurantUpdate(  
  name: String  
  location: String  
  phone: String  
  description: String  
  timings: String  
  cuisine: String  
  delivery_method: String  
  email_id: String  
): Status  
  
createOrder(  
  customer_id: String  
  restaurant_id: String  
  delivery_method: String  
  dish_name: String  
  quantity: Int  
  restaurant_name: String  
): Status  
  
updateOrder(order_id: String, status: String): Status  
  
addReview(  
  rating: String  
  review: String  
  restaurant_id: String  
): Status  
  
addDish(  
  restaurant_id: String  
  name: String  
  ingredients: String  
  price: String  
  category: String  
  description: String  
): Status
```

2. Page showing input & output of a Restaurant Login.

3. Page showing input and output for Adding a dish.

The screenshot shows the GraphQL Playground interface. The URL bar contains the query: `localhost:3001/yelp?query=mutation%7B%0A%09addDish(restaurant_id%3A%5F98f45286232a79557013c5%2C%20name%3A%20Banana%20Milkshake%2C%20ingredients%3A%20Banana%20Milk%2C%20price%3A%201.79%2C%20category%3A%20Beverages%2C%20description%3A%20Its%20a%20shake%2C%20message%3A%20DISH_ADDED%2C%20status%3A%20200%7D)%7B%7D`. The left pane shows the mutation query:

```
1 mutation{
2   addDish(restaurant_id:"5f98f45286232a79557013c5", name:"Banana Milkshake",
3     ingredients:"Banana, Milk", price:"1.79",
4     category:"Beverages", description:"Its a shake."){
5     message
6     status
7   }
8 }
9
```

The right pane shows the JSON response:

```
{
  "data": {
    "addDish": {
      "message": "DISH_ADDED",
      "status": "200"
    }
  }
}
```

The right sidebar shows the schema for the `addDish` mutation:

- No Description**
- TYPE**
- Status**
- ARGUMENTS**
- `restaurant_id: String`
- `name: String`
- `ingredients: String`
- `price: String`
- `category: String`
- `description: String`

4. Page showing input and output for updating any part of the Profile page.

The screenshot shows the GraphQL Playground interface. The URL bar contains the query: `localhost:3001/yelp?query=mutation%7B%0A%09customerUpdate(name%3A%20Khushi%2C%20phone%3A%20123456789%2C%20dob%3A%201990-01-01%2C%20city%3A%20New%20York%2C%20state%3A%20NY%2C%20country%3A%20USA%2C%20nick_name%3A%20Khushi%2C%20about%3A%20I%20love%20coding%2C%20favourite_restaurant%3A%20Sushi%2C%20favourite_hobby%3A%20Reading%2C%20blog_url%3A%20https%3A%2F%2Fexample.com%2F%20blog%2C%20email_id%3A%20khushi%40gundu.com%7D)%7B%7D`. The left pane shows the mutation query:

```
1 mutation{
2   customerUpdate(name:"Khushi", phone:"123456789", dob:"",
3     city:"", state:"", country:"",
4     nick_name:"", about:"", favourite_hobby:"",
5     favourite_restaurant:"", blog_url:"",
6     email_id:"khushi@gundu.com"){
7     message
8     status
9   }
10 }

```

The right pane shows the JSON response:

```
{
  "data": {
    "customerUpdate": {
      "message": "CUSTOMER_UPDATED",
      "status": "200"
    }
  }
}
```

The right sidebar shows the schema for the `customerUpdate` mutation:

- No Description**
- TYPE**
- Status**
- ARGUMENTS**
- `name: String`
- `phone: String`
- `dob: String`
- `city: String`
- `state: String`
- `country: String`
- `nick_name: String`
- `about: String`
- `favourite_restaurant: String`
- `favourite_hobby: String`
- `blog_url: String`
- `email_id: String`



## 7. Page showing input and output for getting a list of orders for a restaurant.

The screenshot shows the GraphQL Playground interface. The query on the left is:

```
1 query{
2   menu(restaurant_id: "5f98f45286232a79557013c5") {
3     name
4     id
5     ingredients
6     description
7     price
8     category
9   }
10 }
```

The JSON response on the right is:

```
{
  "data": {
    "menu": [
      {
        "name": "Pepsi",
        "id": "5fa26a7d56b8f75603dd5ec8",
        "ingredients": "Pepsi drink",
        "description": "Chilled beverage",
        "price": "12",
        "category": "Beverages"
      },
      {
        "name": "Resin Cookies",
        "id": "5fa60a81d85500747b6a510",
        "ingredients": "Resins, Wheat, Oats",
        "description": "Oat Cookies with resins",
        "price": "3.99",
        "category": "Desserts"
      },
      {
        "name": "Banana Milkshake",
        "id": "5fce93dfa0aff1d9e7065a3e",
        "ingredients": "Banana, Milk",
        "description": "Its a shake.",
        "price": "1.79",
        "category": "Beverages"
      },
      {
        "name": "Banana Milkshake",
        "id": "5fce916002e6de1998d7258",
        "ingredients": "Banana, Milk",
        "description": "Its a shake.",
        "price": "1.79",
        "category": "Beverages"
      }
    ]
  }
}
```

The right sidebar shows the schema for the `restaurantOrders` type, which is a list of `Order` objects. The `Order` type has a `restaurant_id` argument of type `String`.

## 8. Page showing input and output for updating delivery status for an order.

The screenshot shows the GraphQL Playground interface. The mutation on the left is:

```
1 mutation{
2   updateOrder(order_id: "5fcd3453d2c00c2d40b0ae1",
3     status: "Order Received"){
4     message
5     status
6   }
7 }
```

The JSON response on the right is:

```
{
  "data": {
    "updateOrder": {
      "message": "ORDER_UPDATED",
      "status": "200"
    }
  }
}
```

The right sidebar shows the schema for the `updateOrder` mutation, which has two arguments: `order_id` of type `String` and `status` of type `String`.

## 5 Enabling multi-part data in GraphQL

For Multi-part data upload using GraphQL, it needs a field called query.

## 6 Architecture for enabling multi-part data in GraphQL

Server side: We need to read the file from the multipart-request, using multer. The file(s) will then be available on the request object, which can be passed into context and processed in the resolve function. The resolve function can upload the file to the the required directory and return the URL.

Client side: We use React Apollo on the client-side to handle GraphQL requests. Here we need to code so that it is capable of accepting multipart requests.

## 7 State any open-source library for enabling multi-part data transfer using GraphQL with sample code. Argue why do you think that this particular library is a good fit.

Multipart is a good open-source library for enabling multi-part data transfer.

```
graphqlServer.use(multipart({
  storage: multipart.memoryStorage(),
}).any());
```

I think this is the best as multipart handles parsing of the raw Http request at the time of file upload.

## 8 Git Commit history

