

Lab 3 Project Report

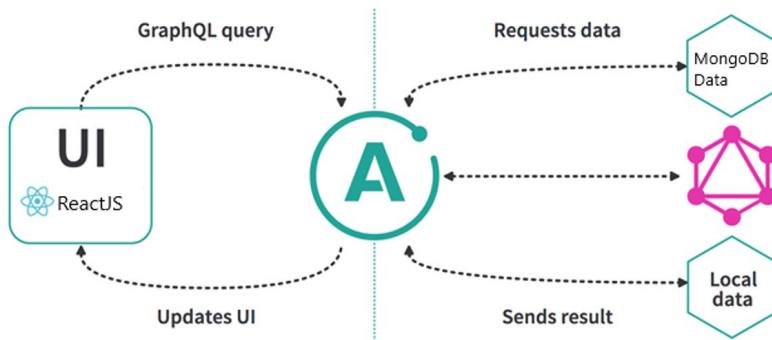
Github Link :

<https://github.com/mathiasjess/Yelp-Lab3-React-GraphQL-MongoDB>

1. Introduction

The objective of this lab is to extend the yelp application built in lab 1 and lab 2 and integrate it with GraphQL. In this lab, the REST API's are replaced by GraphQL. GraphQL in this lab is written as a set of root queries and mutations. Understanding and integrating GraphQL with the existing yelp application is the purpose of this lab. At the frontend react apollo was used to connect and fetch queries and perform mutations from the backend

2. System Design & System Architecture



Frontend: The frontend was designed using react, CSS and redux. This lab had reduced functionalities when compared to lab 2. The react components are mainly divided into restaurant owners and customers. The restaurant owner has child components like restaurant profile details, update restaurant, add menu, view reviews, view orders and update orders. The customer has child components like customer profile, updating the customer profile, searching and displaying restaurant results, viewing a restaurant page, ordering food and writing reviews for a restaurant

The lab 2 API's were replaced by GraphQL root queries and mutations

Backend: The database used was MongoDB. MongoDB had two collections namely

1. Restaurant Owner: Consists of all restaurant details such as basic details, menu items, orders and reviews

2. Customer: This collection has all the basic details of the customer

GraphQL schemas:

1. Restaurant Schema



The screenshot shows a code editor with several tabs at the top: 'schema.js' (highlighted in yellow), 'restaurantGQLSchema.js' (highlighted in green), 'Main.js' (grey), and 'ProfileDetails.js' (grey). The main content area displays the 'restaurantGQLSchema.js' file, which defines a GraphQL schema for a 'RestaurantOwnerType'. The schema includes fields for _id, restaurantName, email, password, description, cuisine, contact, location, city, state, country, zipcode, timings, description, curbPickup, dineIn, yelpDelivery, latitude, longitude, menuItems, events, reviews, and orders. It also imports 'GraphQLObjectType', 'GraphQLString', 'GraphQLBoolean', 'GraphQLFloat', 'GraphQLList', 'GraphQLInt', and 'GraphQLSchema' from 'graphql'.

```
JS schema.js    JS restaurantGQLSchema.js • JS Main.js    JS ProfileDetails.js
Backend > graphQLSchema > JS restaurantGQLSchema.js > (o) ordersType > (o) fields
1 const { GraphQLObjectType,
2   GraphQLString,
3   GraphQLBoolean,
4   GraphQLFloat,
5   GraphQLList,
6   GraphQLInt,
7   GraphQLSchema,
8 } = require('graphql')
9
10 // const restaurant = require('./models/RestaurantOwnerModel')
11
12 const restaurantOwnerType = new GraphQLObjectType({
13   name: "restaurantOwner",
14   fields: () => ({
15     _id: { type: GraphQLString },
16     restaurantName: { type: GraphQLString },
17     email: { type: GraphQLString },
18     password: { type: GraphQLString },
19     description: { type: GraphQLString },
20     cuisine: { type: GraphQLString },
21     contact: { type: GraphQLString },
22     location: { type: GraphQLString },
23     city: { type: GraphQLString },
24     state: { type: GraphQLString },
25     country: { type: GraphQLString },
26     zipcode: { type: GraphQLString },
27     timings: { type: GraphQLString },
28     description: { type: GraphQLString },
29     curbPickup: { type: GraphQLBoolean },
30     dineIn: { type: GraphQLBoolean },
31     yelpDelivery: { type: GraphQLBoolean },
32     latitude: { type: GraphQLFloat },
33     longitude: { type: GraphQLFloat },
34     menuItem: { type: new GraphQLList(menuType) },
35     events: { type: new GraphQLList(eventsType) },
36     reviews: { type: new GraphQLList(reviewsType) },
37     orders: { type: new GraphQLList(ordersType) }
38   })
39 })
```

2. Customer GraphQL schema

```

1  const { GraphQLObjectType,
2   |   GraphQLString,
3   } = require('graphql')
4
5  const customerType = new GraphQLObjectType({
6    name: "customer",
7    fields: () => ({
8      _id : { type: GraphQLString },
9      email : { type: GraphQLString },
10     password : { type: GraphQLString },
11     firstName: { type: GraphQLString },
12     lastName: { type: GraphQLString },
13     DOB : { type: GraphQLString },
14     location :{ type: GraphQLString },
15     city : { type: GraphQLString },
16     state : { type: GraphQLString },
17     country : { type: GraphQLString },
18     zipcode : { type: GraphQLString },
19     nickName: { type: GraphQLString },
20     phoneNumber: { type: GraphQLString },
21     yelpingSince:{ type: GraphQLString },
22     thingsILove: { type: GraphQLString },
23     findMeIn:{ type: GraphQLString },
24     websiteDetails: { type: GraphQLString },
25     profileImage: { type: GraphQLString },
26     favourites: { type: GraphQLString },
27     headline: { type: GraphQLString }
28   })
29 })
30
31 module.exports = customerType

```

3. Questions

1. How will you enable multi-part data in GraphQL?

GraphQL only manages serializable data, and as part of mutations, and hence there we cannot upload files directly. But without using an external library, there are several ways of transferring multipart data, such as:

- Base64 Encoding: The image can be encoded and sent as a string in the Base64 format. This process has many drawbacks, such as
 - The image that is encoded can be larger in size when compared to the original file
 - It is expensive to use encoding and decoding operations

Another option is to request separation during upload: to just have two servers, one for file storage and the other for storing image URL passed as a mutation. The management of several file uploads adds a layer of difficulty. This strategy is not really asynchronous, and the management of the upload server is complex.

2. Discuss the architecture for using multi-part data in GraphQL without using any open source library from Git.

An architecture to use GraphQL multi-part data using no Git open source library.

In order to facilitate file uploads, it is possible to identify the mutations inside the GraphQL schema such that the 64Base encoding of the file string path can be accepted.

Another alternative is to run a separate server only for file uploads. In some applications, this can be achieved. The first file upload request and the second request will use the path of the stored file as a mutation variable.

3. 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?

It will be a good match for our scenario to use the Apollo-upload-server kit (Apollo Server 2.0). Through this package, the client will allow the multipart upload to be sent to the server as a multi-part mutation request. At the backend, this multi-part request is handled by the server and contains an upload agent.

The use of this package is syntactically quite close to our prior upload method using multer.

Sample Code

```
TypeDef code|  
const { ApolloServer, gql } = require('apollo-server');  
const fileuploads = gql`  
type Query {  
  uploads: [File]  
}  
type Mutation {  
  singleUpload(file: Upload!): File!  
}  
`;  
Resolver  
Mutation: {  
  async fileUpload(parent, { file }) {  
    const {filename} = await file;  
  }  
}
```

4. Results with screenshots

1. Restaurant Owner:

Login:

The screenshot shows a web browser window with the URL <http://localhost:3000/login/restaurantlogin>. The page has a red header bar with a logo and navigation links for 'Login' and 'Register'. Below the header is a 'Sign in to Yelp' section with links for 'Existing User?' and 'Restaurant Owner?'. A large circular graphic of a restaurant exterior with people outside is centered. Below the graphic are input fields for email ('cocina@gmail.com') and password ('*****'), and a red 'Restaurant Log In' button.

Restaurant Profile:

The screenshot shows a restaurant profile page for 'Cocina Del Charro'. The page includes a map of San Marcos, Texas, showing the restaurant's location. It displays the restaurant's name, address (1020 W San Marcos Blvd Suite #50), phone number (92078Ph No: 650-897-5437), and a brief description: 'Mexican food from the neighbouring land of Mexico'. It also lists service options: 'Curbside Pickup', 'Yelp Delivery', and 'Dine In'. On the left, there's a sidebar with links for 'View Profile', 'Update Restaurant Profile', 'Add Dishes to Menu', and 'Reviews'. The bottom right corner shows a snippet of the browser's developer tools Network tab.

Update Restaurant Profile:

http://localhost:3000/restauranthomepage/5fb0d020a1e7e5b8c6534d7

Apps Customize AWS Console HeadSpin (6) WhatsApp cURL to Python Importance Of Inter... HeadSpin-Geo Map... Python - Data Stru... Meet - fy-iuar-jgm

Basic Information

Cocina Del Charro

1020 W San Marcos Blvd Suite #50, San Marcos 92078

[View Profile](#) [Update Restaurant Profile](#) [Add Dishes to Menu](#) [Reviews](#)

Restaurant Profile Image Choose File - No file chosen

Restaurant Name Cocina Del Charro

Email cocina@gmail.com

Location 1020 W San Marcos Blvd Suite #50

City San Marcos

State CA

Zip Code 92078

Description Mexican food from the neighbouring land of Mexico

Contact Information
650-423-2341

Timings Mon -Sun : 10AM to 7.30PM

Amenities and more

Curbside Pickup

[Logout](#)

Add Dishes to Menu

http://localhost:3000/restauranthomepage/5fb0d020a1e7e5b8c6534d7

Apps Customize AWS Console HeadSpin (6) WhatsApp cURL to Python Importance Of Inter... HeadSpin-Geo Map... Python - Data Stru... Meet - fy-iuar-jgm

Add dish to restaurant

Cocina Del Charro

1020 W San Marcos Blvd Suite #50, San Marcos 92078

[View Profile](#) [Edit Profile](#) [Update Restaurant Profile](#) [Add Dishes to Menu](#) [Reviews](#)

Choose a category:

Dish Name Fish Fry

Main Ingredients Fish Fry

Description of the Dish Fish Fry

Price 13.99

[Add Dish](#) [Cancel](#)

[Dish Image](#)

Added dish successfully

[Logout](#)

Network Requests Headers Response Headers Request Payload

Name ETag: M75f-2A1A45HAlqNQY07NUCK2
Vary: Origin
X-Powered-By: Express
Request Headers
accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,
Cache-Control: no-cache
Connection: keep-alive
Content-Length: 429
Content-Type: application/json
Host: localhost:3000
Origin: http://localhost:3000
Pragma: no-cache
Referer: http://localhost:3000/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
User-Agent: Mozilla/5.0 (Windows NT
Request Payload
variables: {restaurantId: "5fb0d020a1e7e5b8c6534d7", dishName: "Fish Fry", dishDescription: "Fish Fry", dishPrice: 13.99, dishCategory: "Appetizer"}
query: "mutation addDish(\$restaurantId: String!, \$dishName: String!, \$dishDescription: String!, \$dishPrice: Float!, \$dishCategory: String){ addDish(restaurantId: \$restaurantId, dishName: \$dishName, dishDescription: \$dishDescription, dishPrice: \$dishPrice, dishCategory: \$dishCategory) }"

View Restaurant Dishes

http://localhost:3000/displaymenu

Apps Customize AWS Console HeadSpin (6) WhatsApp cURL to Python Importance Of Inter... HeadSpin-Geo Map... Python - Data Stru... Meet - fy-iuar-jgm

Menu Page

Chicken rolls Category: USA Description: Authentic 6.87 View Details	Chicken rolls Category: USA Description: Authentic 6.87 View Details	Mutton Biryani Category: Main Course Description: Authentic 11.99 View Details
Butter Chicken Category: MainCourse Description: Butter Chicken 10.99 View Details	Fish Fry Category: Appetizer Description: Fish Fry 13.99 View Details	

[Logout](#)

View Reviews given by customers

The screenshot shows a web browser window with the URL <http://localhost:2000/viewcustomerreviews>. The main content area displays a customer review for "Michael Scott" with a rating of 3.7/5, posted on "19-10-2020 11:26:57". The review text is "Comments:Good dish". On the right side, the developer tools Network tab is open, showing a list of requests. One request, named "graphql", is selected, showing its Headers, Preview, Response, and Timing details. The response body contains JSON data related to the review.

Name	Headers	Preview	Response	Timing
graphql	[...]	[...]	[{"id": 1, "customerID": "5f8dc93d657c183ab72fbaa", "customerName": "Michael Scott", "rating": 3.7, "comment": "Good dish", "date": "2020-10-19T11:26:57Z"}]	100 ms 200 ms 300 ms 400 ms 500 ms

View Orders by Customers

All orders

Orders

	Delivery Option:	Status:	Order Type:	Total Price:
 Michael Scott	pickup	Cancelled	Cancelled Order	Update Order Status
 Michael Scott	delivery	Preparing	Delivered	Update Order Status
 Leonard Hofstader	pickup	Preparing	New	Update Order Status
 Sheldon Cooper	pickup	Preparing	Order	Update Order Status

Filters

New Order

Delivered

Canceled Order

Delivery Option: Status: Order Type:
pickup Order Canceled Order

Date: 22-10-2020 11:33:50 Total Price: 0.00

Date: 22-10-2020 11:55:15 Total Price: 45.96

Date: 07-11-2020 12:04:46 Total Price: 17.86

Network tab in browser developer tools showing network requests for the orders.

```
graph TD; A[{"id": "1", "order": "1"}] --> B[{"id": "2", "order": "1"}]; C[{"id": "3", "order": "2"}] --> D[{"id": "4", "order": "2"}]; E[{"id": "5", "order": "3"}] --> F[{"id": "6", "order": "3"}]; G[{"id": "7", "order": "4"}] --> H[{"id": "8", "order": "4"}]
```

Click and view on the profile Page of a customer

The screenshot shows a web browser displaying a customer profile page. The URL is <http://localhost:3000/restaurant/viewCustomer>. The page has a red header bar with links for Home, Menu, Orders, Reviews, and Logout. On the left, there's a sidebar with icons for Apps, Customize, AWS Console, HeadSpin, URL to Python, Importance Of Inte..., HeadSpin-Geo Map..., and Python - LData Stru... (with a progress bar). The main content area features a large photo of a person with glasses and a beard, followed by the customer's name, Leonard Hoffstater (Also known as Leo). Below the name is a headline: HeadLine: #Physics Geek, San Jose, CA, and a note: Favourites include: Star Trek. There are sections for Reviews and About Me, which list location (56 Goulding Street, San Jose, CA, USA, 95132), date of birth (16-02-1987 4:00:00), Yelping Since (06-11-2020 12:13:41), things they love (Facebook, My Blog or Website), and find me in (LinkedIn.com). A developer tools window is open at the bottom right, showing the Network tab with a single request to 'graphql' with a response body of [{"customerDetails": [{"firstname": "Leonard", "lastname": "Hoffstater"}]}]. The status bar at the bottom shows 1 / 2 requests, B66 Line 1, Column 1.

Update delivery status of each customer

Orders

Date: Mon Dec 07 2020 20:22:24 GMT+0800 Total Price: \$45.96

Order details

Mutton Biriyani	11.99	2
Butter Chicken	10.99	2

Delivery Option: delivery **Status:** Preparing **Order Type:** Delivered

Update Order Status

Choose a category:

- On the Way
- Update Order Status**
- Cancel Order

Request Headers:

```
accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,ja;q=0.8
Cache-Control: no-cache
Connection: keep-alive
Content-Length: 145
content-type: application/json
Host: localhost:3000
Origin: http://localhost:3000
Pragma: no-cache
Referer: http://localhost:3000/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36
```

Request Payload:

```
{operationName: "updateorderstatus", variables: {orderID: "5f91d5933454300c28d7d544", status: "Delivered"}, query: "mutation updateorderstatus($orderID: ID!, $status: String!) {order(id: $orderID, status: $status) {id, status}}"} 1 / 2 requests: 401
```

After Updating the orders

All orders

Filters

Michael Scott	Date: 22-10-2020 11:33:50	Total Price: 45.96
Delivery Option: pickup Status: On the Way Order Type: Delivered	Update Order Status	
Leonard Hoffstatter	Date: 07-11-2020 12:04:46	Total Price: 17.86
Delivery Option: pickup Status: Preparing Order Type: New	Update Order Status	

Request Headers:

```
accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,ja;q=0.8
Cache-Control: no-cache
Connection: keep-alive
Content-Length: 145
content-type: application/json
Host: localhost:3000
Origin: http://localhost:3000
Pragma: no-cache
Referer: http://localhost:3000/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36
```

Request Payload:

```
{operationName: "updateorderstatus", variables: {orderID: "5f91d5933454300c28d7d544", status: "Cancelled Order"}, query: "mutation updateorderstatus($orderID: ID!, $status: String!) {order(id: $orderID, status: $status) {id, status}}"} 2 / 14 requests: 401
```

2. Customer

Login Page

Sign in to Yelp

New to Yelp? Sign Up

Existing User? Restaurant Owner?

leonard@gmail.com

User Log In

Once the user logs in, he should be able to see his profile

The screenshot shows a browser window with a red header bar containing various tabs and icons. Below the header is a navigation bar with links like 'Home', 'Messages', 'Events', and 'Users'. The main content area displays a customer profile for 'Leonard Hoffstater (Also known as Leo)'. The profile includes a placeholder profile picture, the name 'Leonard Hoffstater', the alias 'Also known as Leo', and a headline 'HeadLine: #Physics Geek'. It shows the location as 'San Jose, CA' and lists 'Favourites Include: Star Trek'. On the left, there's a sidebar with links for 'Profile Overview', 'Reviews', 'About Me', 'Order History', 'Reviews', 'Friends', 'Review Drafts', 'Compliments', 'Tips', 'Bookmarks', 'Collections', and 'Check-ins'. On the right, there's a 'Network' tab showing network activity with a timeline and a detailed view of a specific request.

Update the customer profile details

This screenshot shows a modal dialog box titled 'localhost:3000 says Updated Profile' over a form for updating customer profile details. The form fields include 'Nick Name' (Cooper), 'Email' (sheilondon@gmail.com), 'Date of Birth' (mm/dd/yyyy), 'Contact Number' (878-654-9208), 'Location' (259 N 1st St), 'City' (San Jose), 'State' (CA), 'Country' (USA), 'Zip Code' (95132), 'HeadLine' (Physics and Mathematics Geek), 'Things I love' (Traveling), 'Favorties' (Star Trek), 'Find me In' (Bing Bang Theory), and 'Blog or website details' (LinkedIn). At the bottom of the modal are 'Save Changes' and 'Cancel' buttons. To the right of the modal is a developer tools Network tab showing a POST request to '/graph' with a status code of 200 OK and a response body containing JSON data related to the update.

The customer can search for a restaurant

The screenshot shows a browser window with a red header bar. The main content area displays a 'Restaurant Search Results' page for 'Cocina Del Charro'. The page includes a map, address (1020 W San Marcos Blvd Suite #50, San Marcos 92078), cuisine (Mexican food from the neighbouring land of Mexico), mode of delivery (Curbside Pickup, Delivery), phone number (600-897-5437), and a 'Visit website' button. To the left, there are filters for 'Mode' (Curbside Pickup, Dine In, Yelp Delivery), 'Curbside Pickup', 'Dine In', and 'Yelp Delivery'. Below these are 'Neighbourhood Locations' and a 'Location' input field. On the right, there's a 'Network' tab in developer tools showing a POST request to '/searchRestaurant' with a status code of 200 OK and a response body containing JSON data.

The customer clicks on the restaurant name to view the restaurant profile page

The screenshot shows a restaurant's customer view page. At the top, there's a header with links for Home, Messages, Events, and Users. Below the header is a map of the area around Cisco Building 10. The main content area has a dark background with the restaurant's name, "Cocina Del Charro", in white. It includes a "Write a review" button, service options (Curbside, Pickup, Yelp Delivery, Dine In), and contact information (Phone: 850-897-5437, Email: cocina@gmail.com, Address: 1020 W San Marcos Blvd Suite #50, San Marcos, CA, 92078). On the left, there's a menu section with two items: Chicken rolls (6.87) and Mutton Biryani (11.99), each with an "Add to Cart" button. The developer tools Network tab is open, showing requests for various resources like 'data:image/png', 'restaurantDetails', and 'graphQL'. One specific request for 'restaurantDetails' is highlighted.

The customer can order food from the restaurant page

This screenshot shows the 'Order Details' page. It lists the selected items: Chicken rolls (6.87) and Mutton Biryani (11.99). The total price is shown as 18.86. There are buttons for 'Add to Cart' and 'Complete Order' or 'Cancel Order'. The developer tools Network tab shows a request for 'restaurantDetails'.

This screenshot shows a confirmation message 'Placed Order Successfully' and the 'Order Food from our Menu' page. The page lists the same items: Chicken rolls (6.87) and Mutton Biryani (11.99). The developer tools Network tab shows a request for 'orderplaced'.

The customer can view his/her order history and filter the orders based on order status and order type (pickup, delivery)

Orders

Filters

Pickup

Delivered

Restaurant: Cocina Del Charro

Order Id: 5f91d08e00373406c626f04

Date: 22-10-2020 11:33:50 **Total Price:**

Delivery Option: pickup **Status:** Cancelled **Order Type:** Cancelled

View Details

Restaurant: Cocina Del Charro

Order Id: 5f91d5833464300c28d7d544

Date: 22-10-2020 11:55:15 **Total Price:** 45.96

Delivery Option: delivery **Status:** Preparing **Order Type:** Delivered

View Details

Restaurant: Cocina Del Charro

Order Id: 8fa6fddc3d4aa07d41aa0ef

Date: 07-11-2020 12:04:46 **Total Price:** 17.86

Delivery Option: pickup **Status:** Preparing **Order Type:** New Order

View Details

Restaurant: Cocina Del Charro

The customer can write a review from the restaurant page

Write a review

Ratings

5.0

Comments

Test Review

Submit Review **Cancel Review**

Added Review

Reviews Page

Ratings: 4.5/5

Dish n Dash Restaurant

Date: 05-11-2020 8:41:13
Comments: Balava good

Ratings: 4.2/5

Thai Orchid

Date: 06-11-2020 4:02:10
Comments: The butter chicken is good

Ratings: 4.5/5

The Cheesecake Factory

Date: 07-12-2020 3:49:00
Comments: Comments from Sheldon Cooper

5. Screenshots of GraphQL Console

1. Page showing list of queries under RootQueryType

The screenshot shows a browser window with the URL `http://localhost:3001/graphiql?query=%0A`. The page title is "RootQueryType". The interface includes a toolbar with various icons like Apps, Customize, AWS Console, HeadSpin, WhatsApp, curl to Python, Importance Of Inte..., HeadSpin-Geo Map..., Python - LData Stru..., and Meet - fy-iue-jgm. Below the toolbar, there are tabs for GraphiQL, Preflight, Merge, Copy, and History. The main area is divided into sections: "FIELDS" containing several query definitions, and "QUERY VARIABLES" which is currently empty. The "FIELDS" section includes:

- restaurantsDetails: [restaurantOwner]
- restaurantDetails(id: String): [restaurantOwner]
- fetchRestaurantOrderSummary(restaurantID: String): [orders]
- customersDetails: [customer]
- customerDetails(id: String): [customer]
- searchRestaurant(searchParameter: String): [restaurantOwner]
- fetchCustomerOrderSummary(customerID: String): [restaurantOwner]
- getCustomerReviews(customerID: String): [restaurantOwner]

JS schema.js X JS Main.js JS ProfileDetails.js JS RestaurantViewOfCustomer.js JS Orders.js

Backend > schema > JS schema.js > [x] RootQuery > fields > customerDetails > resolve > customer.find()

```

46
47 // Root Query
48
49 const RootQuery = new GraphQLObjectType({
50   name: 'RootQueryType',
51   fields: {
52     restaurantDetails: {
53       type: new GraphQLList(restaurantOwnerType),
54       args: { _id: { type: GraphQLString } },
55       resolve(parent, args) {
56         console.log(" Restaurant Id inside restaurant details", args._id)
57         return restaurant.find({ _id: args._id }, (err, result) => {
58           if (err) {
59             throw err
60           }
61           else {
62             // console.log("Restaurant Result", result)
63             return result
64           }
65         })
66       }
67     },
68     fetchRestaurantOrderSummary: {
69       type: new GraphQLList(ordersType),
70       args: { restaurantID: { type: GraphQLString } },
71       async resolve(parent, args) {
72         return await fetchRestaurantOrderSummary(args)
73       }
74     },
75     customerDetails: {
76       type: new GraphQLList(customerType),
77       args: { _id: { type: GraphQLString } },
78       resolve(parent, args) {
79         return customer.find({ _id: args._id }, (err, result) => {
80           if (err) {
81             throw err
82           }
83           else {
84             console.log("Customer Result")
85             return result
86           }
87         })
88       }
89     },
90   }
91 }
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114

```

JS schema.js X JS Main.js JS ProfileDetails.js JS RestaurantViewOfCustomer.js JS Orders.js

Backend > schema > JS schema.js > [x] RootQuery > fields > customerDetails > searchRestaurant

```

90   searchRestaurant: {
91     type: new GraphQLList(restaurantOwnerType),
92     args: { searchParameter: { type: GraphQLString } },
93     async resolve(parent, args) {
94       return await searchForRestaurant(args)
95     }
96   },
97   fetchCustomerOrderSummary: {
98     type: new GraphQLList(restaurantOwnerType),
99     args: { customerID: { type: GraphQLString } },
100    async resolve(parent, args) {
101      return await fetchCustomerOrderSummary(args)
102    }
103  },
104  getcustomerReviews: {
105    type: new GraphQLList(restaurantOwnerType),
106    args: { customerID: { type: GraphQLString } },
107    async resolve(parent, args) {
108      console.log("Customer ID", args.customerID)
109      return await getCustomerReviews(args)
110    }
111  }
112}
113
114

```

2. List of mutations under Mutation.

The screenshot shows the GraphQL playground interface at <http://localhost:3001/graphiql?query=%0A>. The left sidebar has tabs for GraphiQL, Prettify, Merge, Copy, and History. The right panel is titled 'Mutation' and contains a search bar and a 'No Description' message. Below that is a 'FIELDS' section with two mutation definitions:

```
restaurantRegistration(  
    restaurantName: String  
    email: String  
    password: String  
    location: String  
    city: String  
    state: String  
    country: String  
    zipcode: String  
): Status  
  
restaurantLogin(email: String, password: String): Status
```

```
updateRestaurant(  
    restaurantId: String  
    restaurantName: String  
    email: String  
    description: String  
    contact: String  
    timings: String  
    curbPickup: Boolean  
    dineIn: Boolean  
    yelpDelivery: Boolean  
    location: String  
    city: String  
    state: String  
    country: String  
    zipcode: String  
): Status
```

The screenshot shows the GraphQL playground interface at <http://localhost:3001/graphiql?query=%0A>. The left sidebar has tabs for GraphiQL, Prettify, Merge, Copy, and History. The right panel is titled 'Mutation' and contains a search bar and a 'No Description' message. Below that is a 'FIELDS' section with two mutation definitions:

```
addDish(  
    restaurantId: String  
    dishName: String  
    dishIngredients: String  
    dishDescription: String  
    price: String  
    dishCategory: String  
): Status
```

```
customerRegistration(  
    firstName: String  
    lastName: String  
    email: String  
    password: String  
): Status  
  
customerLogin(email: String, password: String): Status
```

```
updateCustomer(  
    customerId: String  
    email: String  
    firstName: String  
    lastName: String  
    DOB: String  
    location: String  
    city: String  
    state: String  
    country: String  
    nickName: String  
    phoneNumber: String  
    thingsILove: String  
    findMeIn: String  
    websiteDetails: String  
    favourites: String  
    headline: String  
    zipcode: String  
): Status
```

← → ⌂ ⌂ http://localhost:3001/graphql?query=%0A

Apps Customise AWS Console HeadSpin (6) WhatsApp cURL to Python CT Importance Of Inte... HeadSpin-Geo Map... Python - LData Stru... Meet - fy-tuer-jgm

Update

GraphIQL Prettify Merge Copy History

< Schema

```
customerID: String
email: String
firstName: String
lastName: String
DOB: String
location: String
city: String
state: String
country: String
nickName: String
phoneNumber: String
thingsLove: String
findmein: String
websiteDetails: String
favcolor: String
headline: String
zipcode: String
); Status
```

writeReviews(

```
restaurantID: String
customerID: String
customerName: String
ratings: String
comments: String
); Status
```

placeOrder(

```
restaurantID: String
customerID: String
customerName: String
totalPrice: String
deliveryOption: String
deliveryStatus: String
deliveryFilter: String
orderDetails: String
); Status
```

updateOrderStatus(

```
orderID: String
deliveryStatus: String
deliveryFilter: String
); Status
```

QUERY VARIABLES

GraphQL schema for mutations

Backend > schema > [JS schema.js](#) > [RootQuery](#) > [fields](#) > [customerDetails](#) > [resolve](#)

```
115 const Mutation = new GraphQLObjectType({
116   name: "Mutation",
117   fields: {
118     restaurantRegistration: {
119       type: statusType,
120       args: {
121         restaurantName: { type: GraphQLString },
122         email: { type: GraphQLString },
123         password: { type: GraphQLString },
124         location: { type: GraphQLString },
125         city: { type: GraphQLString },
126         state: { type: GraphQLString },
127         country: { type: GraphQLString },
128         zipcode: { type: GraphQLString },
129       },
130       async resolve(parent, args) {
131         return await restaurantRegister(args)
132       }
133     },
134     restaurantLogin: {
135       type: statusType,
136       args: {
137         email: { type: GraphQLString },
138         password: { type: GraphQLString }
139       },
140       async resolve(parent, args) {
141         return await restaurantLogin(args)
142       }
143     },
144     updateRestaurant: {
145       type: statusType,
146       args: {
147         restaurantId: { type: GraphQLString },
148         restaurantName: { type: GraphQLString },
149         email: { type: GraphQLString },
150         description: { type: GraphQLString },
151         contact: { type: GraphQLString },
152         timings: { type: GraphQLString },
153         curbPickup: { type: GraphQLBoolean },
154         dineIn: { type: GraphQLBoolean },
155         yelpDelivery: { type: GraphQLBoolean },
156         location: { type: GraphQLString },
157         city: { type: GraphQLString },
158         state: { type: GraphQLString },
159       },
160       async resolve(parent, args) {
161         return await updateRestaurantProfile(args)
162       }
163     },
164     addDish: {
165       type: statusType,
166       args: {
167         restaurantId: { type: GraphQLString },
168         dishName: { type: GraphQLString },
169         dishIngredients: { type: GraphQLString },
170         dishDescription: { type: GraphQLString },
171         price: { type: GraphQLString },
172         dishCategory: { type: GraphQLString },
173       },
174       async resolve(parent, args) {
175         console.log("Inside Add Dish", args.restaurantId)
176         return await addMenu(args)
177       }
178     },
179     customerRegistration: {
180       type: statusType,
181       args: {
182         firstName: { type: GraphQLString },
183         lastName: { type: GraphQLString },
184         email: { type: GraphQLString },
185         password: { type: GraphQLString },
186       },
187       async resolve(parent, args) {
188         return await customerRegister(args)
189       }
190     },
191     customerLogin: {
192       type: statusType,
193       args: {
194         email: { type: GraphQLString },
195         password: { type: GraphQLString }
196       },
197       async resolve(parent, args) {
198         return await customerLogin(args)
199       }
200     },
201   },
202 }
```

The image shows a code editor with two tabs open, both titled "schema.js". The left tab contains the following GraphQL schema code:

```
Backend > schema > JS schema.js > [e] RootQuery > ⚡ fields > ⚡ customerDetails > ⚡ resolve > ⚡ updateCustomer: {  
  type: statusType,  
  args: {  
    customerId: { type: GraphQLString },  
    email: { type: GraphQLString },  
    firstName: { type: GraphQLString },  
    lastName: { type: GraphQLString },  
    DOB: { type: GraphQLString },  
    location: { type: GraphQLString },  
    city: { type: GraphQLString },  
    state: { type: GraphQLString },  
    country: { type: GraphQLString },  
    nickName: { type: GraphQLString },  
    phoneNumber: { type: GraphQLString },  
    thingsILove: { type: GraphQLString },  
    findMeIn: { type: GraphQLString },  
    websiteDetails: { type: GraphQLString },  
    favourites: { type: GraphQLString },  
    headline: { type: GraphQLString },  
    zipcode: { type: GraphQLString }  
  },  
  async resolve(parent, args) {  
    console.log("Update Customer Profile", args.customerId)  
    return await updateCustomerProfile(args)  
  }  
},  
writeReviews: {  
  type: statusType,  
  args: {  
    restaurantID: { type: GraphQLString },  
    customerID: { type: GraphQLString },  
    customerName: { type: GraphQLString },  
    ratings: { type: GraphQLString },  
    comments: { type: GraphQLString }  
  },  
  async resolve(parent, args) {  
    return await writeReview(args)  
  }  
},
```

The right tab contains the following GraphQL schema code:

```
Backend > schema > JS schema.js > [e] RootQuery > ⚡ fields > ⚡ customerDetails > ⚡ placeOrder: {  
  type: statusType,  
  args: {  
    restaurantID: { type: GraphQLString },  
    customerID: { type: GraphQLString },  
    customerName: { type: GraphQLString },  
    totalPrice: { type: GraphQLString },  
    deliveryOption: { type: GraphQLString },  
    deliveryStatus: { type: GraphQLString },  
    deliveryFilter: { type: GraphQLString },  
    orderDetails: { type: GraphQLString }  
  },  
  async resolve(parent, args) {  
    console.log("Order details", args.orderDetails)  
    return await placeOrder(args)  
  }  
},  
updateOrderStatus: {  
  type: statusType,  
  args: {  
    orderID: { type: GraphQLString },  
    deliveryStatus: { type: GraphQLString },  
    deliveryFilter: { type: GraphQLString }  
  },  
  async resolve(parent, args) {  
    return await updateOrderStatus(args)  
  }  
},  
module.exports = new GraphQLSchema({  
  query: RootQuery,  
  mutation: Mutation  
})
```

3. Page showing input & output of a Restaurant Login

The screenshot shows the GraphQL playground interface at <http://localhost:3001/graphqI>. The left panel displays a GraphQL mutation query:

```
1 mutation{
2   restaurantLogin(email:"cocina@gmail.com",password: "cocina@123"){
3     status
4     message
5   }
6 }
7
```

The right panel shows the results of the mutation:

```
+ {
+   "data": {
+     "restaurantLogin": {
+       "status": "200",
+       "message": "Success"
+     }
+   }
+ }
```

On the far right, there is a detailed view of the mutation:

- Mutation:** `restaurantLogin`
- Type:** No Description
- Status:** Status
- Arguments:**
 - `email: String`
 - `password: String`

4. Page showing input and output for Adding a dish

The screenshot shows the GraphQL playground interface on a web browser. The URL is [http://localhost:3001/graphql?query=mutation%7B%0A%20%20addDish\(restaurantId%3A%5fa838692ce2f40d07190755%2CdishName%3A%20Prawn%20Pulao%2C%20dish...](http://localhost:3001/graphql?query=mutation%7B%0A%20%20addDish(restaurantId%3A%5fa838692ce2f40d07190755%2CdishName%3A%20Prawn%20Pulao%2C%20dish...). The left panel contains the GraphQL query:

```

1+ mutation{
2+   addDish(restaurantId:"5fa838692ce2f40d07190755",dishName: "Prawn Pulao", dishIngredient: "status"
3+   message
4+ }
5+
6+
7+

```

The right panel shows the mutation details:

- Mutation:** addDish
- Type:** No Description
- Status:** TYPE
- Arguments:**
 - restaurantId: String
 - dishName: String
 - dishIngredients: String
 - dishDescription: String
 - price: String
 - dishCategory: String

Output:

The screenshot shows the GraphQL playground interface on a web browser. The URL is [http://localhost:3001/graphql?query=query%7B%0A%20%20%20restaurantDetails\(_id%3A%5fa838692ce2f40d07190755\)%7B%0A%20%20%20%20menuItem{...%7D%0A%20%20%20%20...](http://localhost:3001/graphql?query=query%7B%0A%20%20%20restaurantDetails(_id%3A%5fa838692ce2f40d07190755)%7B%0A%20%20%20%20menuItem{...%7D%0A%20%20%20%20...). The left panel contains the GraphQL query:

```

1+ query{
2+   restaurantDetails(_id:"5fa838692ce2f40d07190755"){
3+     menuItem{
4+       dishName,
5+       dishCategory,
6+       dishDescription,
7+       dishIngredients,
8+       price
9+     }
10+   }
11+
12+

```

The right panel shows the query results:

- RootQueryType:** restaurantDetails
- Type:** No Description
- Arguments:** {_id: String}
- Results:**
 - Object 1: dishName: "Chicken with Pita Bread", dishCategory: "Salads", dishDescription: "Chicken with Pita Bread", dishIngredients: "Chicken with Pita Bread", price: "9.19"
 - Object 2: dishName: "Rice and beans", dishCategory: "Salads", dishDescription: "Rice and beans", dishIngredients: "Rice and beans", price: "7.49"
 - Object 3: dishName: "Shrimp Rolls", dishCategory: "Appetizer", dishDescription: "Shrimp Rolls", dishIngredients: "Shrimp Rolls", price: "6.99"
 - Object 4: dishName: "Mixed Platter", dishCategory: "Deserts", dishDescription: "Mixed Platter", dishIngredients: "Mixed Platter", price: "7.99"
 - Object 5: dishName: "Red Velvet Cheese Cake", dishCategory: "Deserts", dishDescription: "Red Velvet Cheese Cake", dishIngredients: "Red Velvet Cheese Cake", price: "7.99"
 - Object 6: dishName: "Caesar Salad", dishCategory: "Salads", dishDescription: "Caesar Salad", dishIngredients: "Caesar Salad", price: "8.99"
 - Object 7: dishName: "Prawn Pulao", dishCategory: "Main Course", dishDescription: "Delicious Indian Dish", dishIngredients: "Prawns, Rice", price: "9.99"

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

The screenshot shows the GraphQL playground interface on a browser. The URL is `http://localhost:3001/graphQL?query=mutation%7B%0A%20updateRestaurant(restaurantId%3A"5fa838692ce2f40d07190755"%2CrestaurantName%3A%20The%20Cheesecake...%0A%20)%7D`. The main area displays a GraphQL mutation named `updateRestaurant` with its schema and variables.

Mutation: `updateRestaurant`

No Description

TYPE: Status

ARGUMENTS:

- restaurantId: String
- restaurantName: String
- email: String
- description: String
- contact: String
- timings: String
- currPickup: Boolean
- dineIn: Boolean
- yelpDelivery: Boolean
- location: String
- city: String
- state: String
- country: String
- zipcode: String

QUERY VARIABLES:

```
1 mutation
2 updateRestaurant(restaurantId:"5fa838692ce2f40d07190755",restaurantName: "The Cheesecake"
3   , dineIn:false, yelpDelivery: true, location:"1631 N Capitol Ave", city:"San Jose", s
4   tatus
5   message
6
7 }
8 }
```

```
+ {
  "data": [
    "updateRestaurant": {
      "status": "200",
      "message": "Updated Profile"
    }
  ]
}
```

Output:

6. Page showing input and output for Restaurant Search

The screenshot shows the GraphQL playground interface at <http://localhost:3001/graphiql>. The left panel contains the GraphQL query:

```

1+ query{
2+   searchRestaurant(searchParameter:"The Cheesecake Factory"){
3+     restaurantName,
4+     description,
5+     location,
6+     city,
7+     zipcode,
8+     timings,
9+     contact,
10+    curbPickup,
11+    yelpDelivery,
12+    dineIn
13+  }
14+}
15+

```

The right panel shows the response schema for the `searchRestaurant` mutation. It includes fields for `data`, `searchRestaurant`, and a nested object with details like `restaurantName`, `description`, `location`, `city`, `zipcode`, `timings`, `contact`, `curbPickup`, `yelpDelivery`, and `dineIn`.

TYPE: `[RestaurantOwner]`

ARGUMENTS: `searchParameter: String`

7. Page showing input and output for placing an order.

The screenshot shows the GraphQL playground interface at <http://localhost:3001/graphiql>. The left panel contains the GraphQL mutation:

```

1+ mutation{
2+   placeOrder(restaurantId:"5fa838692ce2f40d07190755", customerID: "5f990fb56841a558781b718f",
3+   customerName:"Jennifer Anniston", totalPrice:"22.99",
4+   deliveryOption:"pickup", delivery_status:"Order Received", deliveryFilter:"New Order",
5+   orderDetails:"Manchurian,1,7.99,Noodles,2, 15.00"){
6+     status,
7+     message
8+   }
9+ }
10+

```

The right panel shows the response schema for the `placeOrder` mutation. It includes fields for `data`, `placeOrder`, and a nested object with `status` and `message` fields.

TYPE: `Status`

ARGUMENTS:

- `restaurantId: String`
- `customerID: String`
- `customerName: String`
- `totalPrice: String`
- `deliveryOption: String`
- `delivery_status: String`
- `deliveryFilter: String`
- `orderDetails: String`

Output:

The screenshot shows the GraphQL playground interface. On the left, a code editor displays a query to get restaurant details by ID. On the right, a mutation named "placeOrder" is defined with fields for customer ID, total price, delivery option, delivery filter, and order details.

```

1+ query{
2+   restaurantDetails(_id:"5fa838692ce2f40d07190755"){
3+     orders{
4+       orderDate,
5+       customerID,
6+       customerName,
7+       deliveryOption,
8+       deliveryFilter,
9+       delivery_status
10+      orderDetails{
11+        dishName,
12+        quantity,
13+        price
14+      }
15+    }
16+  }
17+
}

```

Mutation:

```

placeOrder {
  TYPE
  Status
  ARGUMENTS
  restaurantId: String!
  customerID: String!
  customerName: String!
  totalPrice: String!
  deliveryOption: String!
  deliveryStatus: String!
  deliveryFilter: String!
  orderDetails: String!
}

```

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

The screenshot shows the GraphQL playground interface. A complex query is displayed, starting with "query" and ending with "orderDetails". It includes fields for restaurant details, orders, and order details, which further nest into dish names, quantities, and prices.

```

1+ query{
2+   restaurantDetails(_id:"5fa838692ce2f40d07190755"){
3+     restaurantName
4+     description
5+     orders{
6+       orderDate,
7+       customerID,
8+       customerName,
9+       deliveryOption,
10+      deliveryFilter,
11+      delivery_status
12+      orderDetails{
13+        dishName,
14+        quantity,
15+        price
16+      }
17+    }
18+  }
19+
}

```

RootQueryType:

```

restaurantsDetails {
  No Description
  TYPE
  [restaurantOwner]
}

```

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

The screenshot shows the GraphQL playground interface at <http://localhost:3001/graphq>. The left panel contains the following GraphQL mutation:

```

1+ mutation{
2+   updateOrderStatus(orderID:"5fceda695e17904e40f395e8",
3+     delivery_status:"Pick up ready",deliveryFilter:"New Order"){
4+   status,
5+   message
6+ }
7+
}
  
```

The right panel shows the response schema for the mutation:

```

{
  "data": {
    "updateOrderStatus": {
      "status": "200",
      "message": "Updated Order Status"
    }
  }
}
  
```

The Documentation Explorer sidebar on the right provides information about the schema:

- Search Schema...**
- A GraphQL schema provides a root type for each kind of operation.
- ROOT TYPES**
- query:** RootQueryType
- mutation:** Mutation

Output:

The screenshot shows the GraphQL playground interface at <http://localhost:3001/graphq>. The left panel contains the following GraphQL query:

```

1+ query{
2+   restaurantDetails(_id:"5fa838692ce2f40d07190755"){
3+     restaurantName,
4+     orders{
5+       _id,
6+       customerID,
7+       customerName,
8+       deliveryOption,
9+       deliveryFilter,
10+      delivery_status
11+    }
12+  }
13+
}
  
```

The right panel shows the response schema for the query:

```

{
  "data": {
    "restaurantDetails": [
      {
        "restaurantName": "The Cheesecake Factory",
        "orders": [
          {
            "_id": "5fa864c09aaed81ac640d5b6",
            "customerID": "5fa840762ce2f40d0719076e",
            "customerName": "Darth Vader",
            "deliveryOption": "delivery",
            "deliveryFilter": "New Order",
            "delivery_status": "Preparing"
          },
          {
            "_id": "5fceda695e17904e40f395e8",
            "customerID": "5f900fb56841a558781b718f",
            "customerName": "Jennifer Anniston",
            "deliveryOption": "pickup",
            "deliveryFilter": "New Order",
            "delivery_status": "Pick up ready"
          }
        ]
      }
    ]
  }
}
  
```

The Documentation Explorer sidebar on the right provides information about the mutation:

- No Description**
- TYPE**
- Status**
- ARGUMENTS**
- orderID:** String
- delivery_status:** String
- deliveryFilter:** String

10. Write a customer review

The screenshot shows the GraphQL playground interface. On the left, the query editor contains the following GraphQL mutation:

```

1+ mutation{
2+   writeReviews(restaurantId:"5fa838692ce2f40d07190755", customerId:"5f990fb56841a558781b718f",
3+     customerName:"Jennifer Anniston", ratings:"4.5", comments:" the restaurant has a good view"
4+   )
5+   status,
6+   message
7+ }
  
```

The results pane on the right shows the response from the server:

```

{
  "data": {
    "writeReviews": {
      "status": "200",
      "message": "Added Review"
    }
  }
}
  
```

Details on the right side of the interface include:

- Mutation:** writeReviews
- TYPE:** No Description
- ARGS:**
 - restaurantId: String
 - customerId: String
 - customerName: String
 - ratings: String
 - comments: String

11. Get customer reviews

The screenshot shows the GraphQL playground interface. On the left, the query editor contains the following GraphQL query:

```

1+ query{
2+   getCustomerReviews(customerID:"5f990fb56841a558781b718f"){
3+     restaurantName,
4+     reviews{
5+       customerId,
6+       customerName,
7+       reviewDate,
8+       ratings,
9+       comments
10+    }
11+ }
  
```

The results pane on the right shows the response from the server:

```

{
  "data": {
    "getCustomerReviews": [
      {
        "restaurantName": "The Cheesecake Factory",
        "reviews": [
          {
            "customerId": "5fa19d18445a13129cfbc9e7",
            "customerName": "Sheldon Cooper",
            "reviewDate": "1607384990189",
            "ratings": 4.5,
            "comments": "Comments from Sheldon Cooper"
          },
          {
            "customerId": "5fa19d18445a13129cfbc9e7",
            "customerName": "Sheldon Cooper",
            "reviewDate": "1607384990189",
            "ratings": 4.5,
            "comments": "Comments from Sheldon Cooper"
          },
          {
            "customerId": "5f990fb56841a558781b718f",
            "customerName": "Jennifer Anniston",
            "reviewDate": "1607403465692",
            "ratings": 4.5,
            "comments": " the restaurant has a good view"
          },
          {
            "customerId": "5f990fb56841a558781b718f",
            "customerName": "Jennifer Anniston",
            "reviewDate": "1607403465692",
            "ratings": 4.5,
            "comments": " the restaurant has a good view"
          }
        ]
      }
    ]
  }
}
  
```

Details on the right side of the interface include:

- RootQueryType:** getCustomerReview
- TYPE:** No Description
- ARGS:** customerID: String

7. Screenshots of Github Repo

The screenshot shows the GitHub repository page for 'Yelp-Lab3-React-GraphQL-MongoDB'. The repository is private and has 1 branch and 0 tags. The main branch contains 13 commits from the user 'mathiasjess'. The commits are listed as follows:

- Added customer profile view for restaurant (34 seconds ago)
- Backend: Added customer profile view for restaurant (34 seconds ago)
- Frontend: Added customer profile view for restaurant (34 seconds ago)
- .gitattributes: Initial commit (16 days ago)
- README.md: Update README.md (16 days ago)

The repository also includes a README.md file with the following content:

```
Yelp-Lab3-react-node-garphqk-mongodb

Prototype of yelp application using react node and mysql
```

On the right side of the page, there are sections for About, Releases, Packages, and Languages. The Languages section shows that the code is primarily written in JavaScript (89.4%), CSS (9.9%), and HTML (0.7%).

The screenshot shows the commit history for the 'main' branch. The commits are organized by date:

- Commits on Dec 7, 2020:
 - optimized the front end to display graphql data (mathiasjess committed 1 minute ago)
 - Added order history and write review for customer (mathiasjess committed 4 hours ago)
 - added front end code to place orders (mathiasjess committed 19 hours ago)
 - added frontend for customer login, profile and search (mathiasjess committed 20 hours ago)
- Commits on Dec 6, 2020:
 - added front end graphql code for adding menu, reviews and orders (mathiasjess committed yesterday)
 - Implemented front end graphql with apollo client for restaurant regis... (mathiasjess committed 2 days ago)
- Commits on Nov 24, 2020:
 - Deleted routes for backend (mathiasjess committed 13 days ago)
 - Implemented backend logic for restaurant and customer (mathiasjess committed 13 days ago)

Commits on Dec 6, 2020

- added front end graphql code for adding menu, reviews and orders
mathiasjess committed yesterday
- Implemented front end graphql with apollo client for restaurant regis...
mathiasjess committed 2 days ago

Commits on Nov 24, 2020

- Deleted routes for backend
mathiasjess committed 13 days ago
- Implemented backend logic for restaurant and customer
mathiasjess committed 13 days ago

Commits on Nov 23, 2020

- GraphQL backend for Restaurant Registration and login
mathiasjess committed 14 days ago

Commits on Nov 21, 2020

- Update README.md
mathiasjess committed 16 days ago
- Initial commit
mathiasjess committed 16 days ago
- Initial commit
mathiasjess committed 16 days ago

Newer Older

Read the guide

mathiasjess / Yelp-Lab3-React-GraphQL-MongoDB Private

Code Issues Pull requests Actions Projects Security Insights Settings

Who has access

PRIVATE REPOSITORY Only those with access to this repository can view it.

DIRECT ACCESS 1 has access to this repository. 1 invitation.

Manage

Manage access

Select all Type ▾

Find a collaborator...

Puneetjyot Awaiting puneetjyot's response Pending Invite

← Previous Next →