Jessica Mathias
ID: 013838097

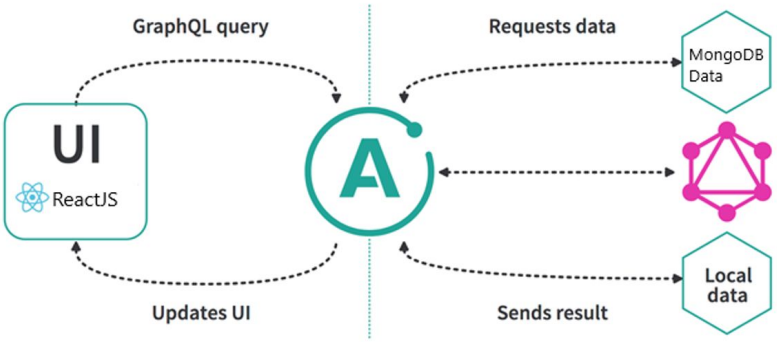
Lab 3
Project Report

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

2. Customer GraphQL schema

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.

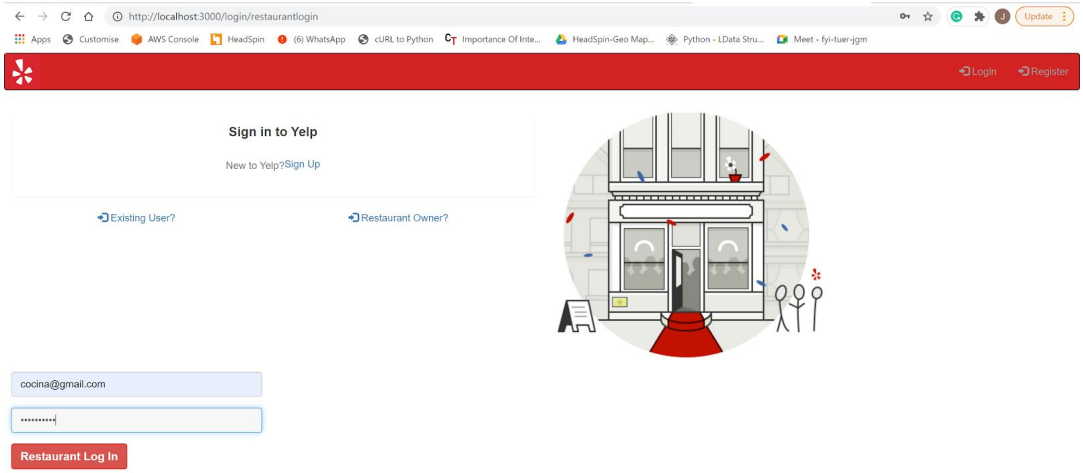
Sample Code

```
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;
}
```

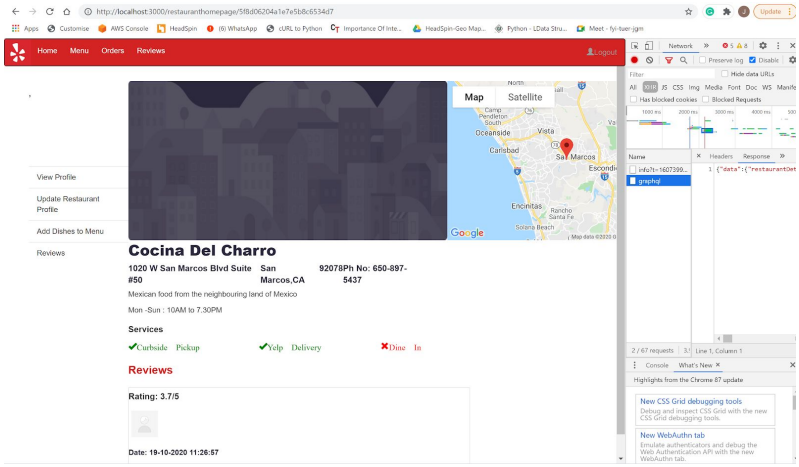
4. **Results with screenshots**

1. **Restaurant Owner:**

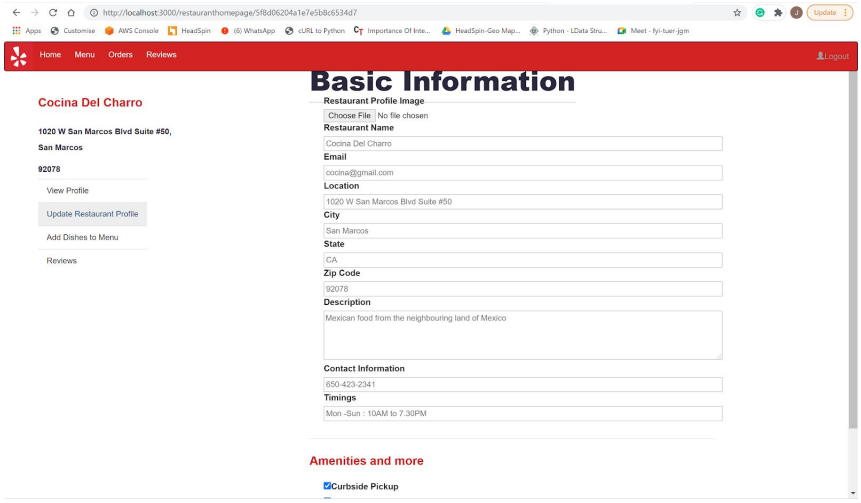
Login:



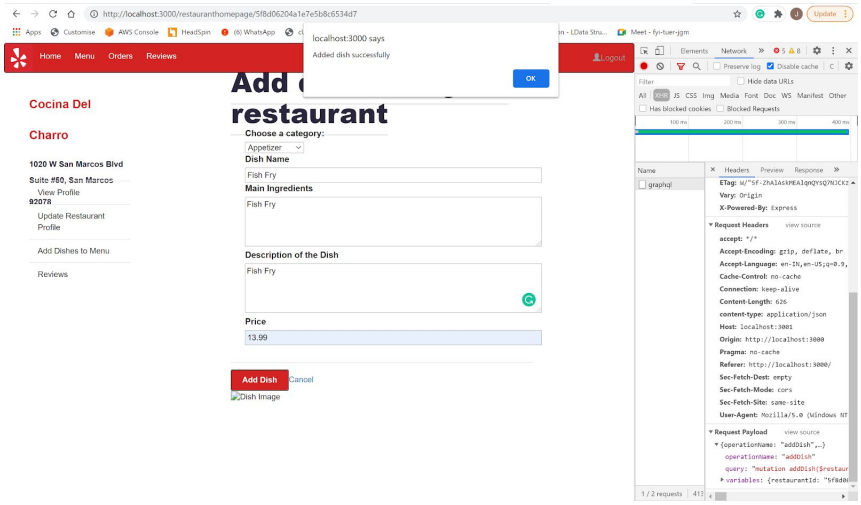
Restaurant Profile:



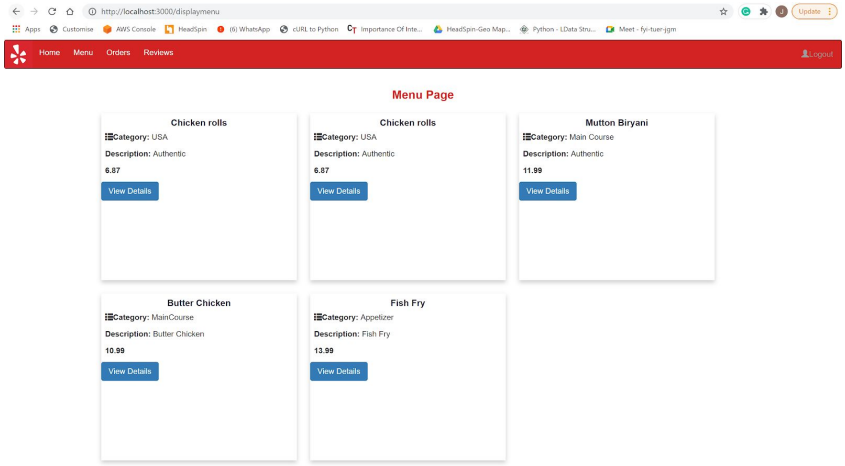
Update Restaurant Profile:



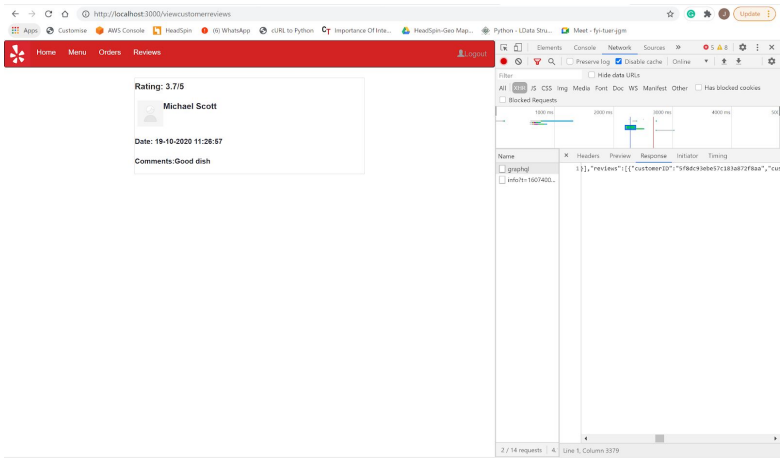
Add Dishes to Menu



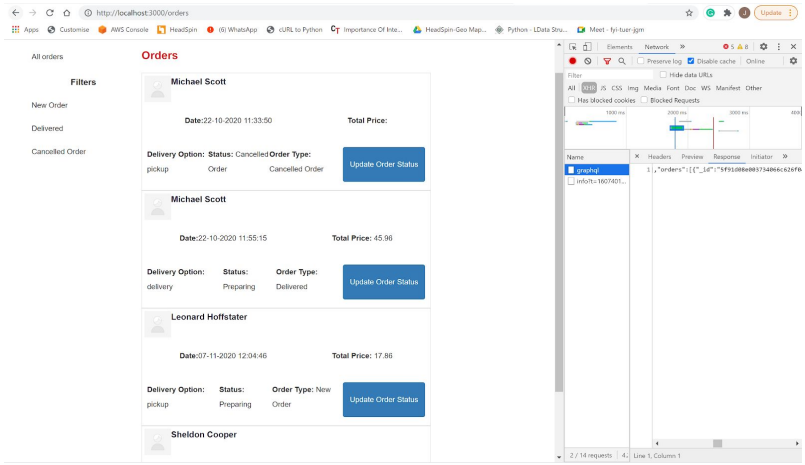
View Restaurant Dishes



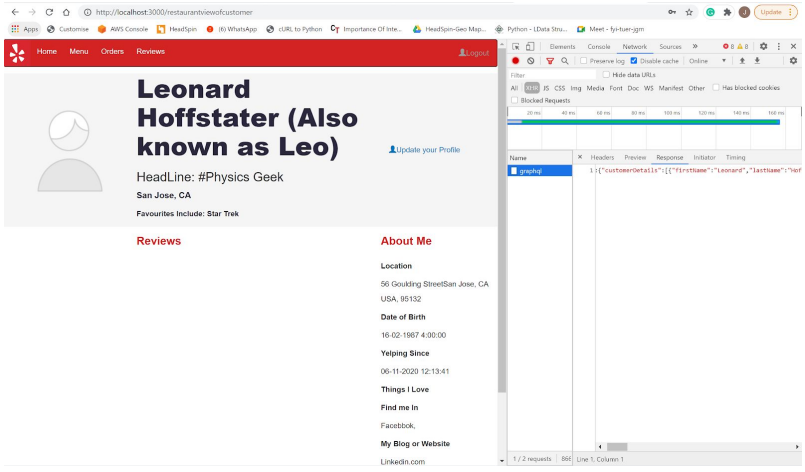
View Reviews given by customers



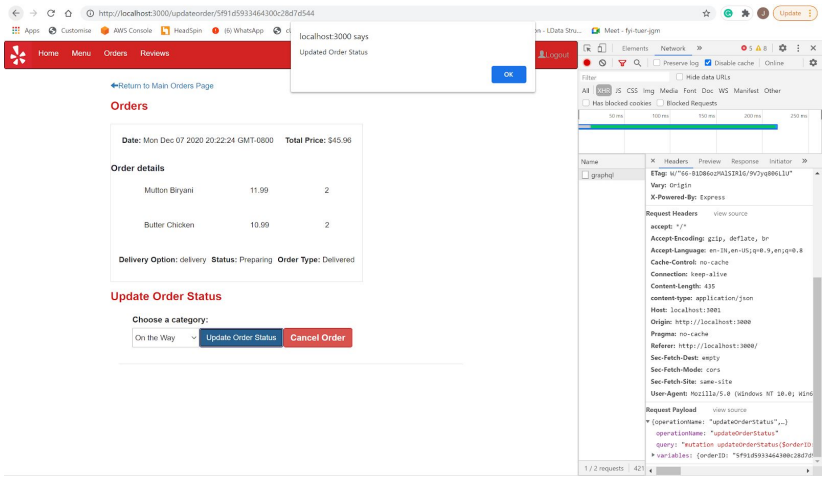
View Orders by Customers



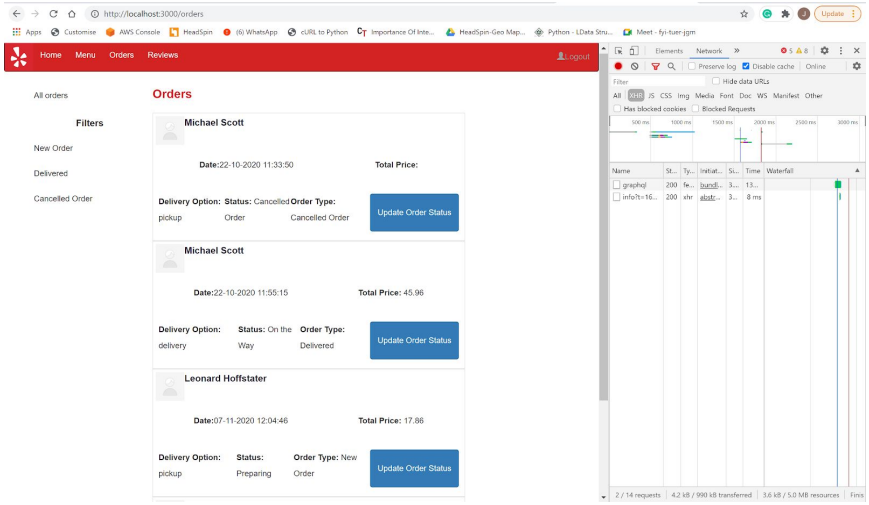
Click and view on the profile Page of a customer



Update delivery status of each customer

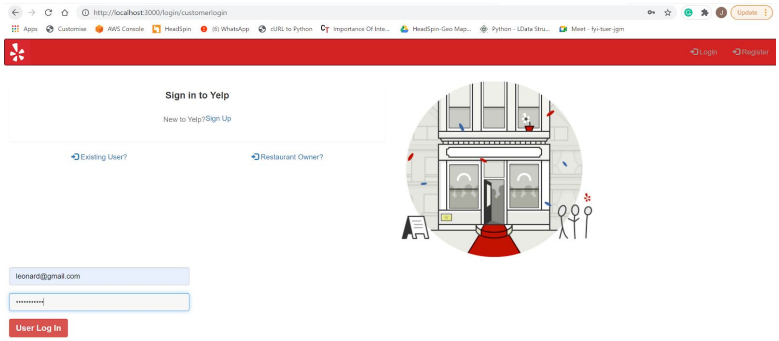


After Updating the orders

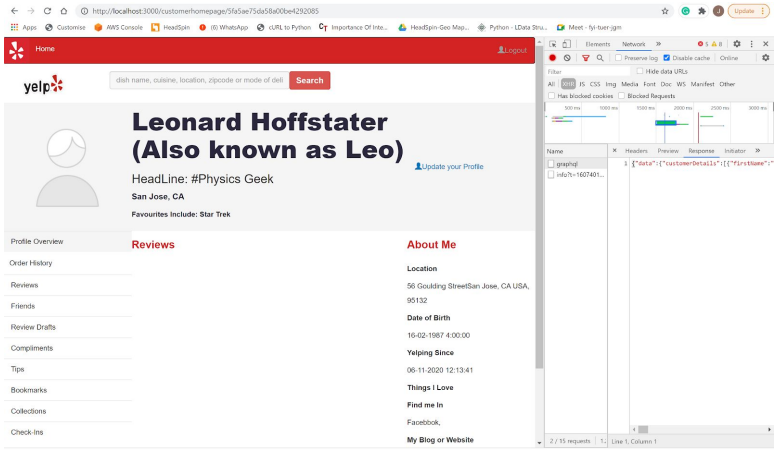


2. Customer

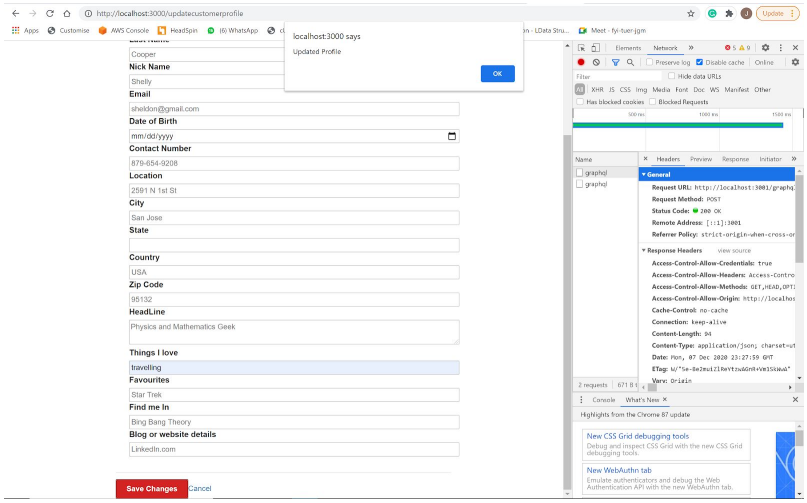
Login Page



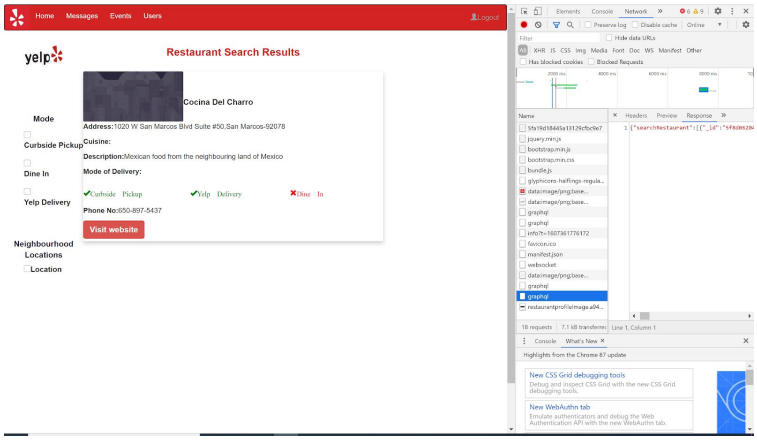
Once the user logs in, he should be able to see his profile



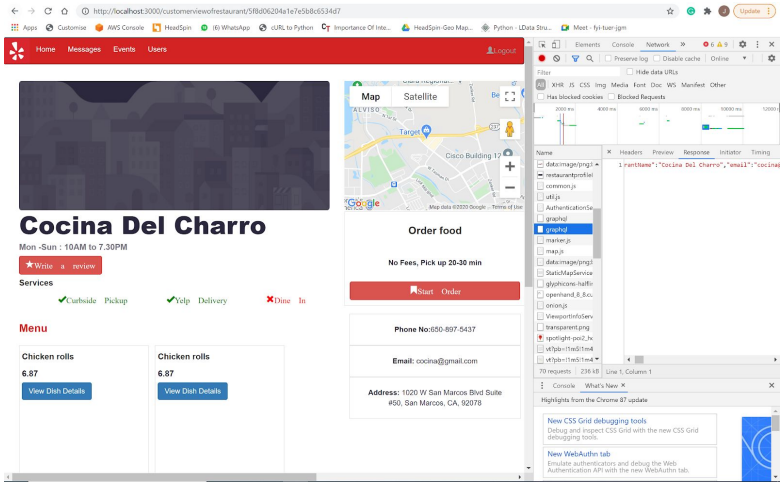
Update the customer profile details



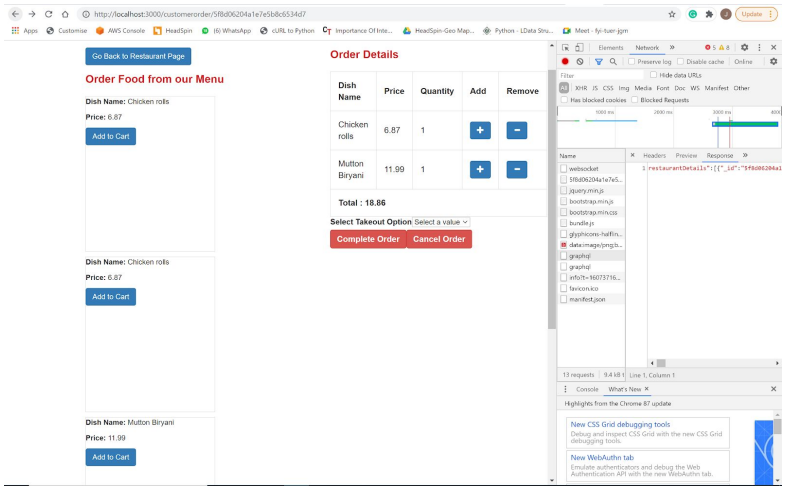
The customer can search for a restaurant

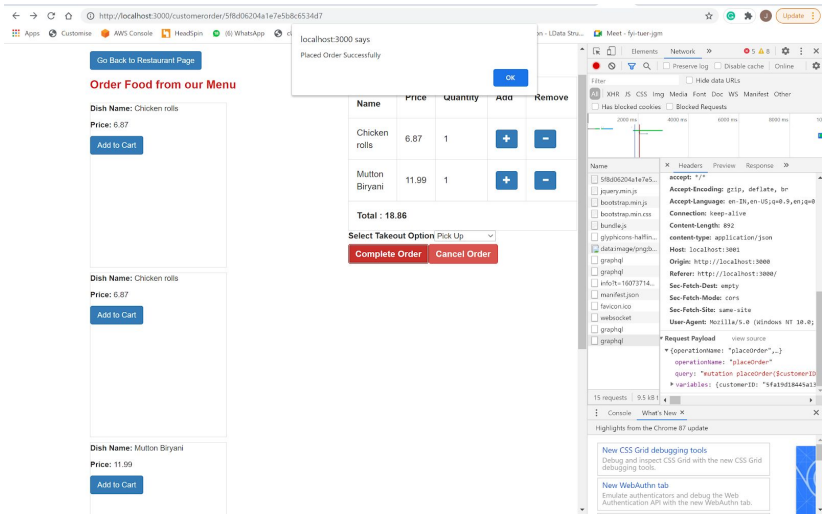


The customer clicks on the restaurant name to view the restaurant profile page

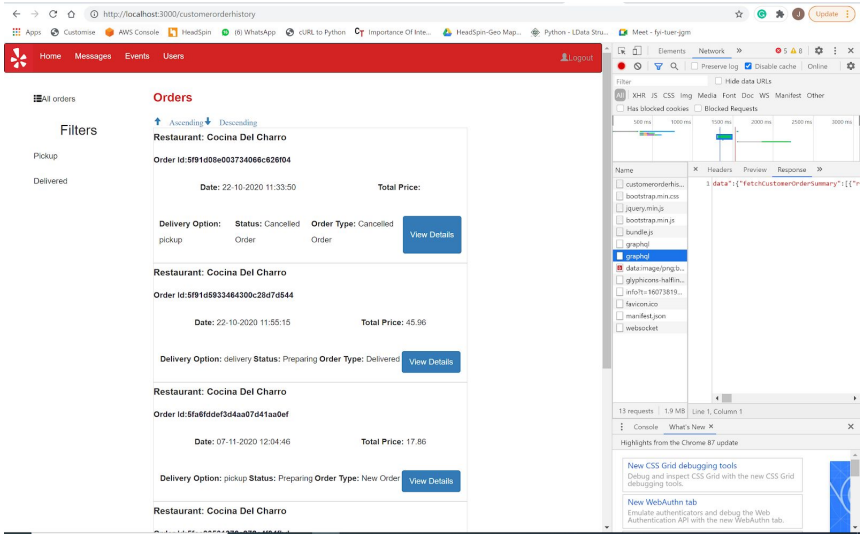


The customer can order food from the restaurant page

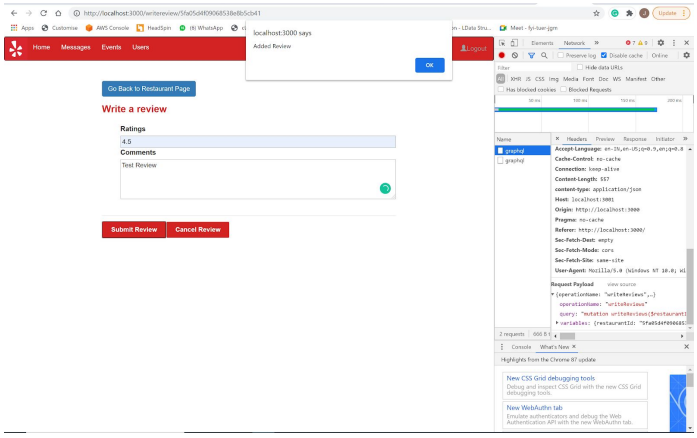


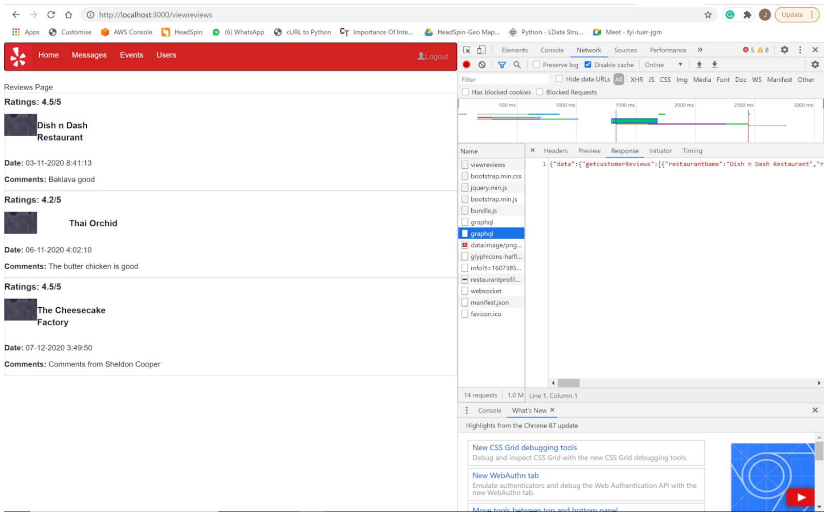


The customer can view his/her order history and filter the orders based on order status and order type (pickup, delivery)



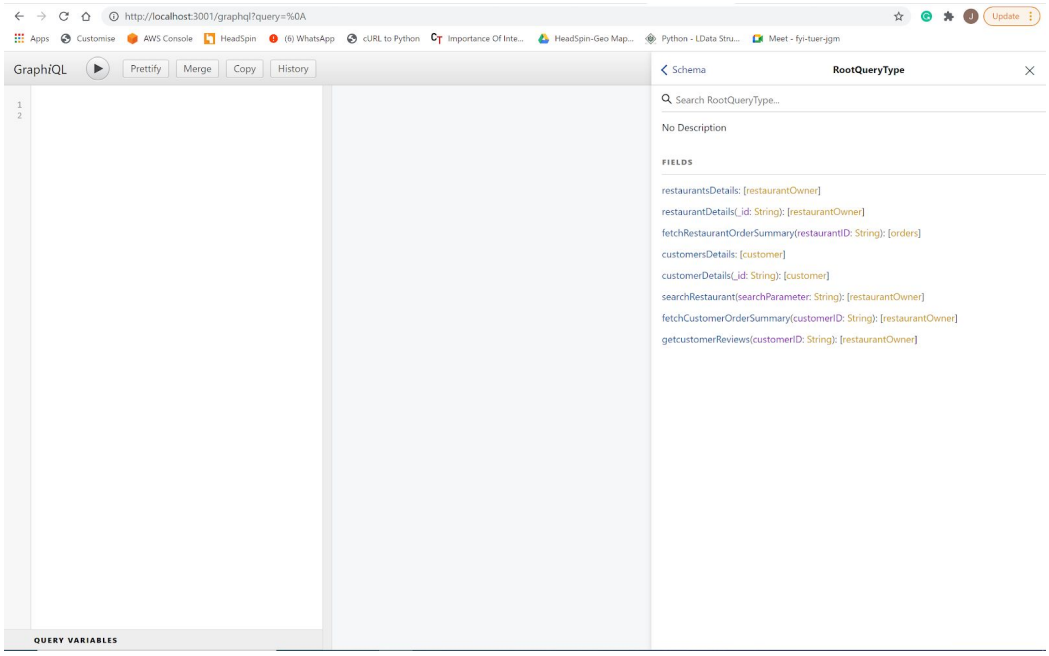
The customer can write a review from the restaurant page





5. Screenshots of GraphQL Console

1. Page showing list of queries under RootQueryType



schema.js | Main.js | ProfileDetails.js | RestaurantViewOfCustomer.js | Orders.js

Backend > schema > schema.js > RootQuery > fields > customerDetails > resolve > customer.find() c

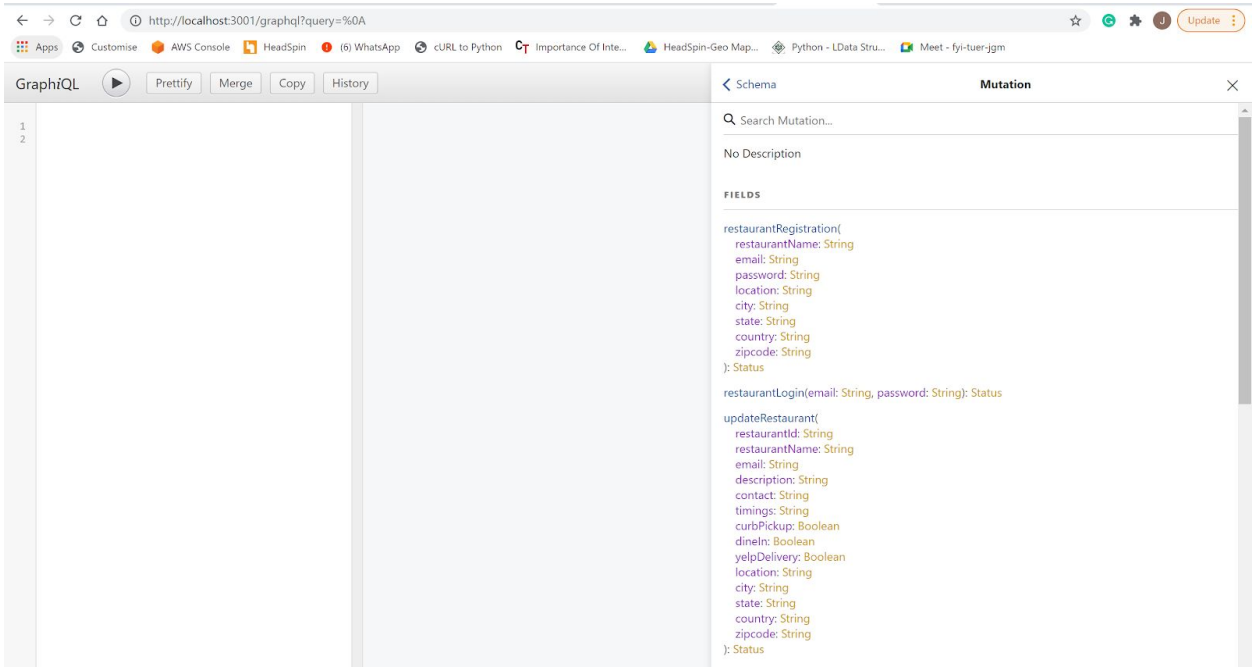
```
46
47  // Root Query
48
49  const RootQuery = new GraphQLObjectType({
50      name: 'RootQueryType',
51      fields: {
52          restaurantDetails: {
53              type: new GraphQLList(restaurantOwnerType),
54              args: { _id: { type: GraphQLString } },
55              resolve(parent, 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          },
```

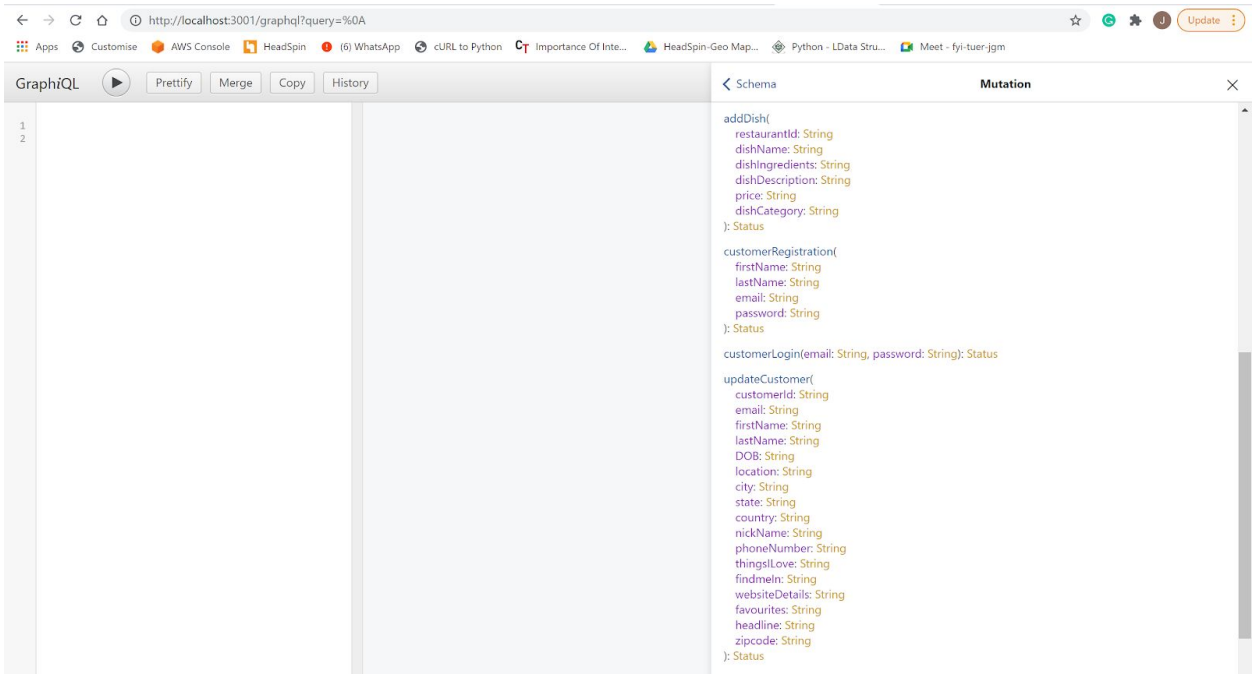
schema.js | Main.js | ProfileDetails.js | RestaurantViewOfCust

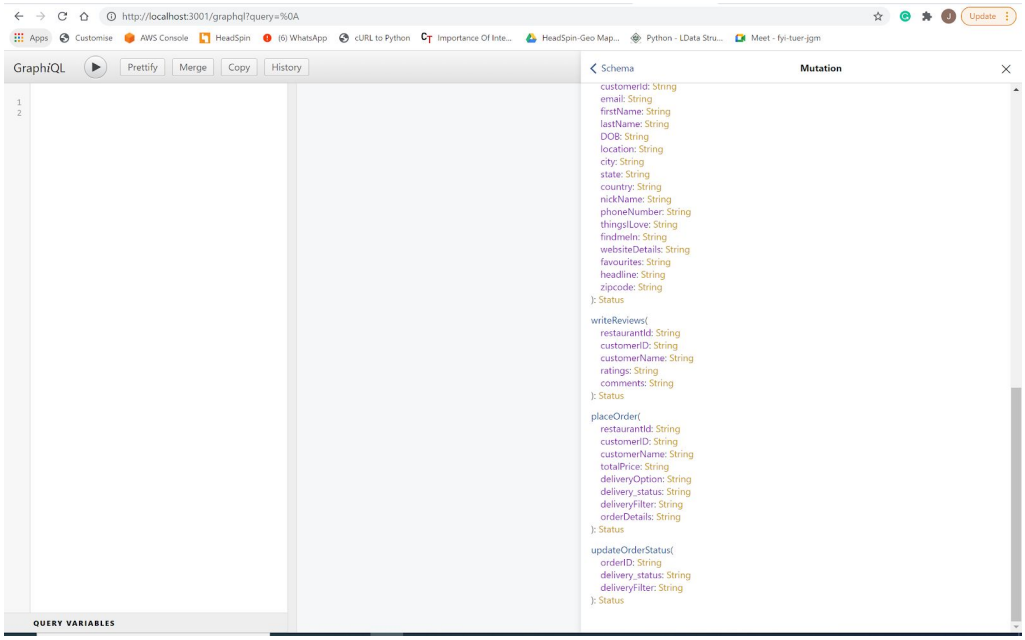
Backend > schema > schema.js > RootQuery > fields > customerDetails >

```
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.







GraphQL schema for mutations

schema.js | Main.js | ProfileDetails.js | RestaurantViewOfCustomer

Backend > schema > schema.js > RootQuery > fields > customerDetails > reso

```
const Mutation = new GraphQLObjectType({
    name: "Mutation",
    fields: {
        restaurantRegistration: {
            type: statusType,
            args: {
                restaurantName: { type: GraphQLString },
                email: { type: GraphQLString },
                password: { type: GraphQLString },
                location: { type: GraphQLString },
                city: { type: GraphQLString },
                state: { type: GraphQLString },
                country: { type: GraphQLString },
                zipcode: { type: GraphQLString },
            },
            async resolve(parent, args) {
                return await restaurantRegister(args)
            }
        },
        restaurantLogin: {
            type: statusType,
            args: {
                email: { type: GraphQLString },
                password: { type: GraphQLString }
            },
            async resolve(parent, args) {
                return await restaurantLogin(args)
            }
        },
        updateRestaurant: {
            type: statusType,
            args: {
                restaurantId: { type: GraphQLString },
                restaurantName: { type: GraphQLString },
                email: { type: GraphQLString },
                description: { type: GraphQLString },
                contact: { type: GraphQLString },
                timings: { type: GraphQLString },
                curbPickup: { type: GraphQLBoolean },
                dineIn: { type: GraphQLBoolean },
                yelpDelivery: { type: GraphQLBoolean },
                location: { type: GraphQLString },
                city: { type: GraphQLString },
                state: { type: GraphQLString },
```

schema.js | Main.js | ProfileDetails.js | RestaurantViewOfCustomer.js

Backend > schema > schema.js > RootQuery > fields > customerDetails > resolve

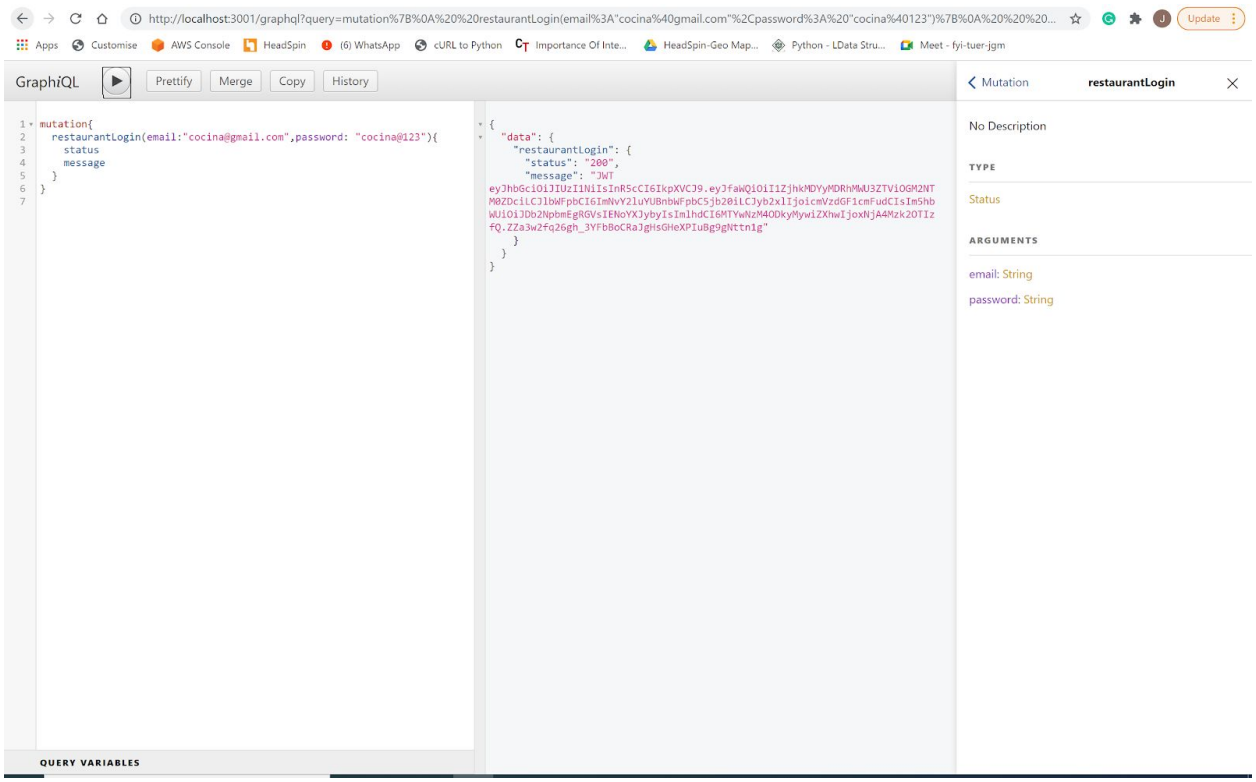
```
                country: { type: GraphQLString },
                zipcode: { type: GraphQLString },
            },
            async resolve(parent, args) {
                return await updateRestaurantProfile(args)
            }
        },
        addDish: {
            type: statusType,
            args: {
                restaurantId: { type: GraphQLString },
                dishName: { type: GraphQLString },
                dishIngredients: { type: GraphQLString },
                dishDescription: { type: GraphQLString },
                price: { type: GraphQLString },
                dishCategory: { type: GraphQLString },
            },
            async resolve(parent, args) {
                console.log("Inside Add Dish", args.restaurantId)
                return await addMenu(args)
            }
        },
        customerRegistration: {
            type: statusType,
            args: {
                firstName: { type: GraphQLString },
                lastName: { type: GraphQLString },
                email: { type: GraphQLString },
                password: { type: GraphQLString },
            },
            async resolve(parent, args) {
                return await customerRegister(args)
            }
        },
        customerLogin: {
            type: statusType,
            args: {
                email: { type: GraphQLString },
                password: { type: GraphQLString }
            },
            async resolve(parent, args) {
                return await customerLogin(args)
            }
        },
```

```
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)
    }
},
```

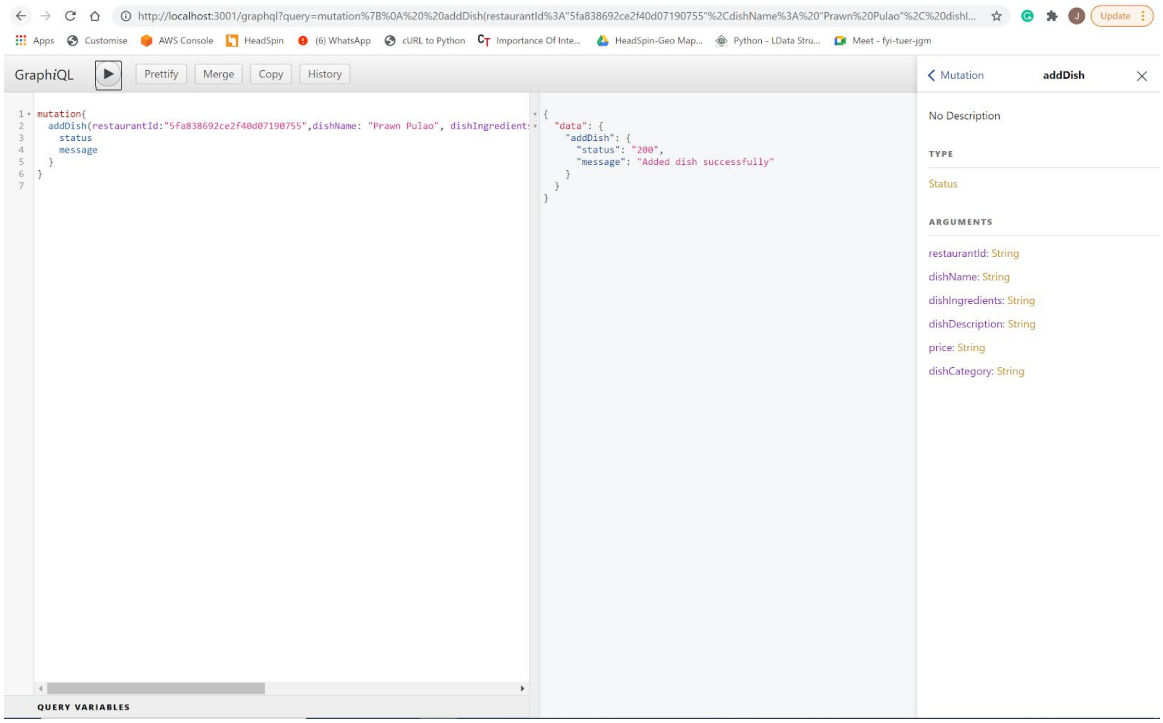
```
placeOrder: {
    type: statusType,
    args: {
        restaurantId: { type: GraphQLString },
        customerID: { type: GraphQLString },
        customerName: { type: GraphQLString },
        totalPrice: { type: GraphQLString },
        deliveryOption: { type: GraphQLString },
        delivery_status: { 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 },
        delivery_status: { type: GraphQLString },
        deliveryFilter: { type: GraphQLString }
    },
    async resolve(parent, args) {
        return await updateOrderStatus(args)
    }
},
}
})

module.exports = new GraphQLSchema({
    query: RootQuery,
    mutation: Mutation
})
```

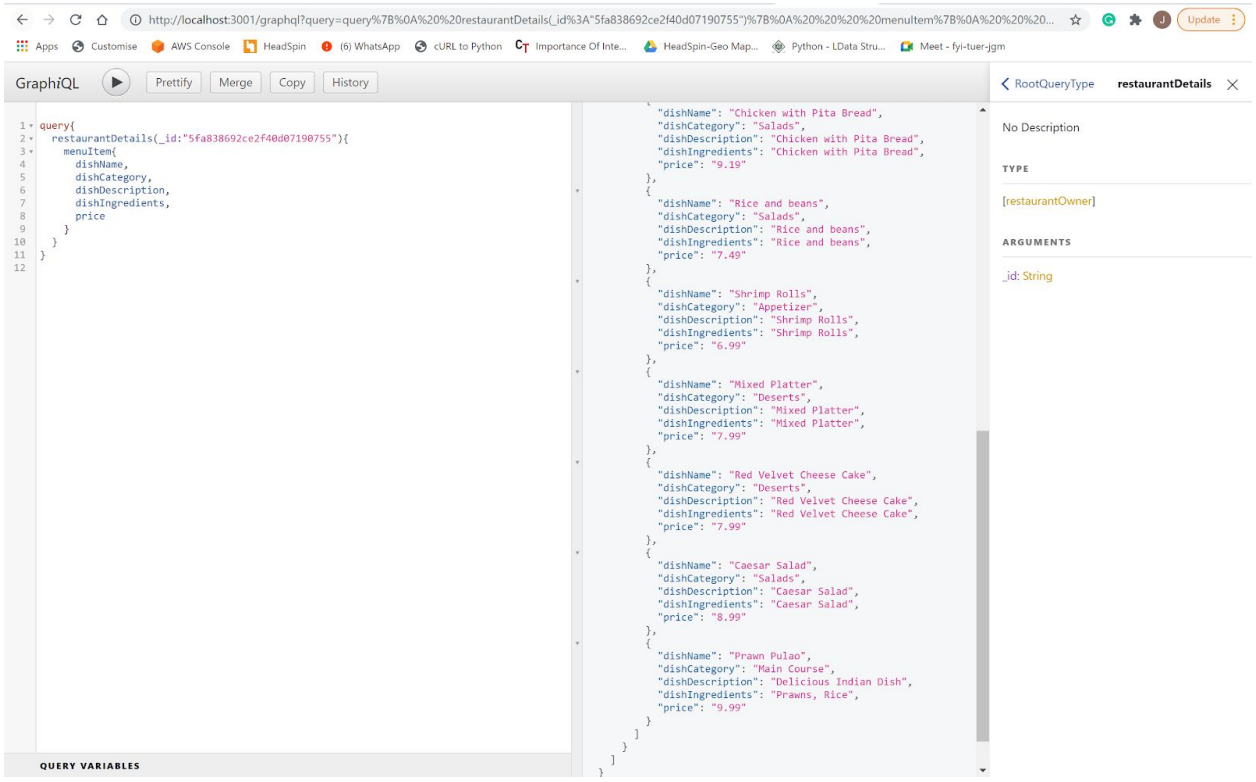
3. Page showing input & output of a Restaurant Login



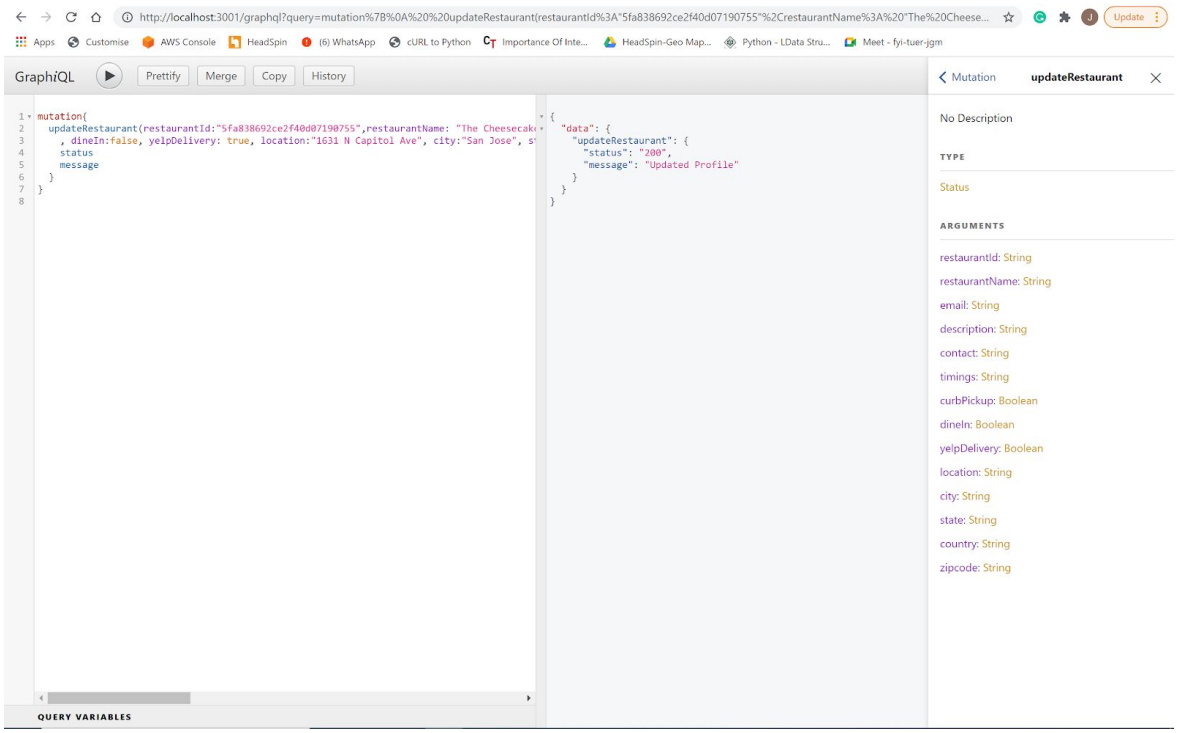
4. Page showing input and output for Adding a dish



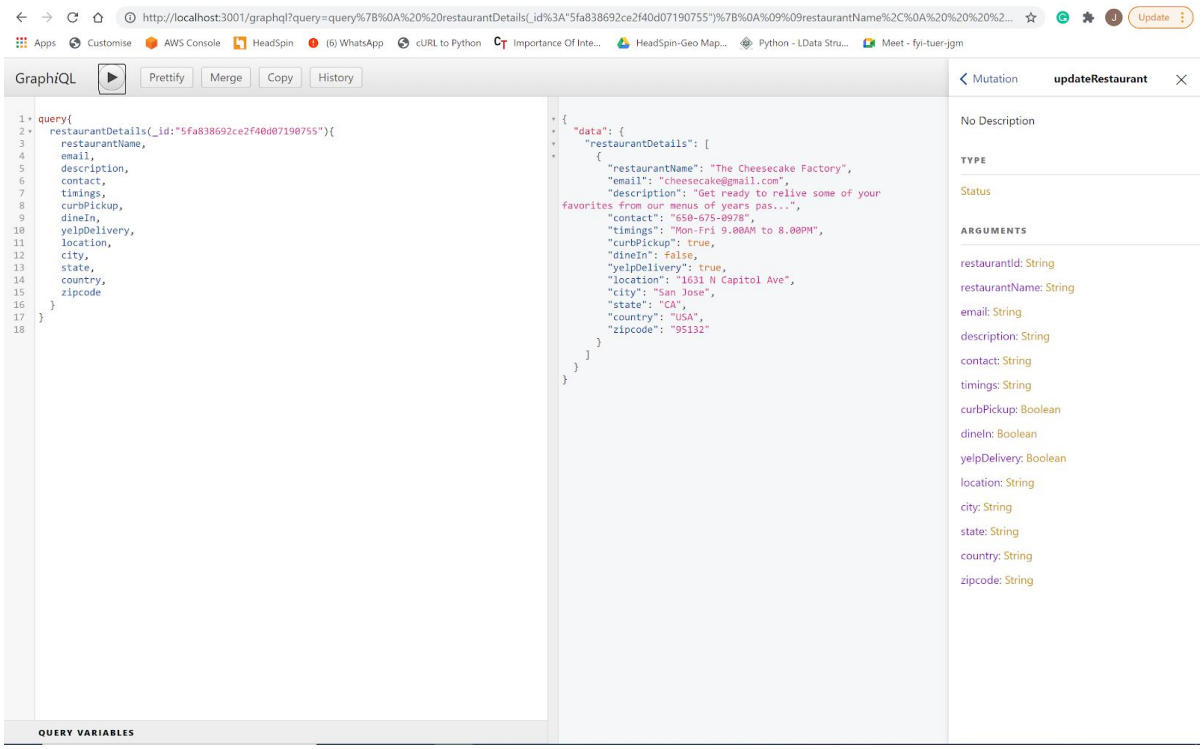
Output:



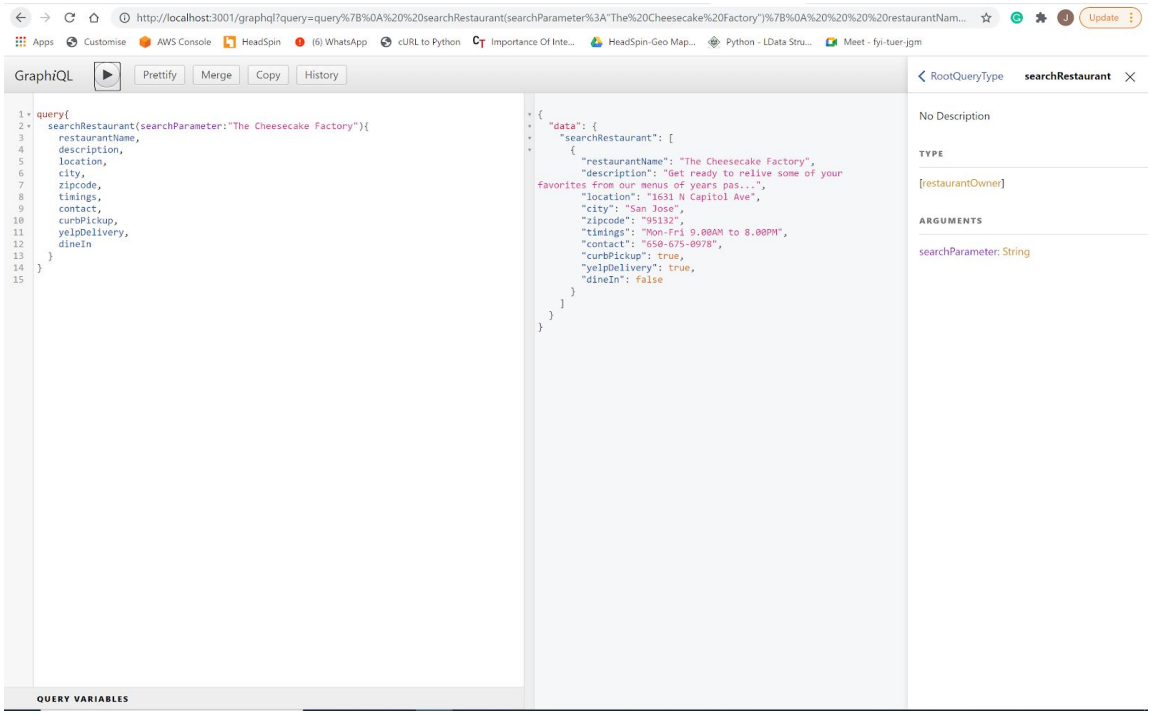
5. Page showing input and output for updating any part of the Profile page



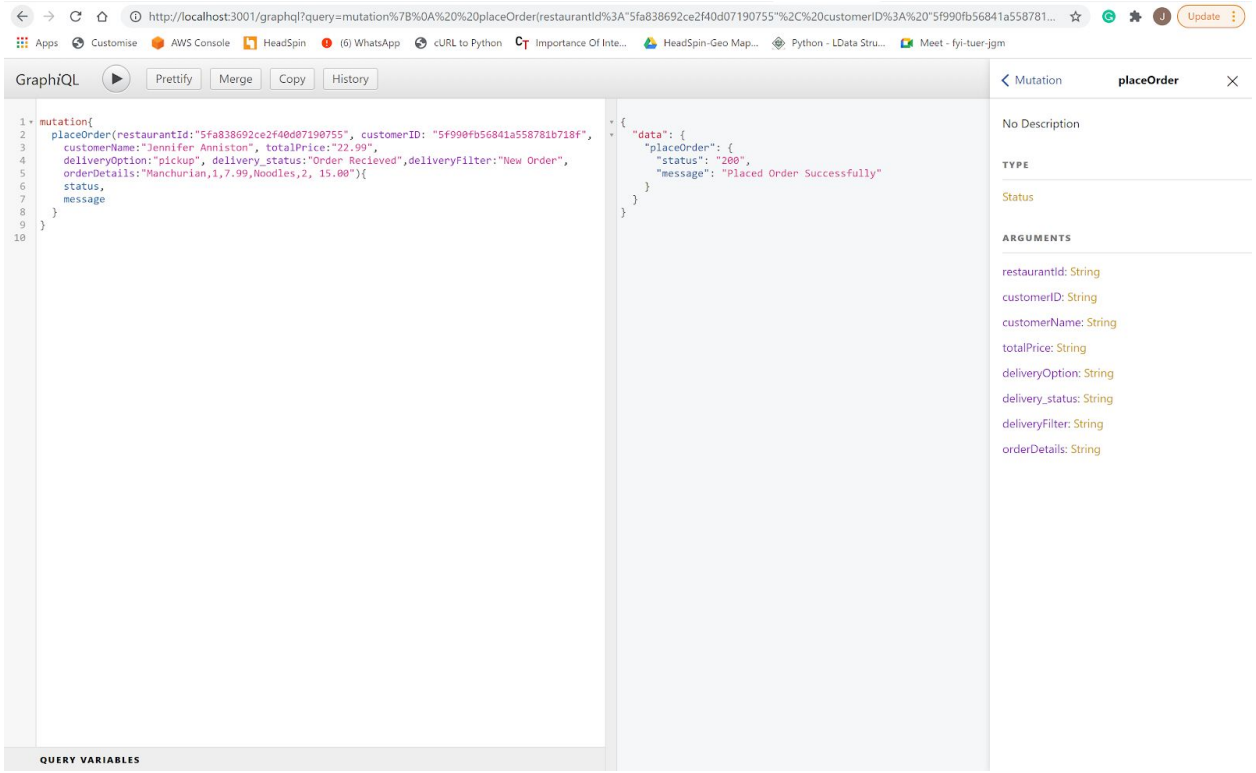
Output:



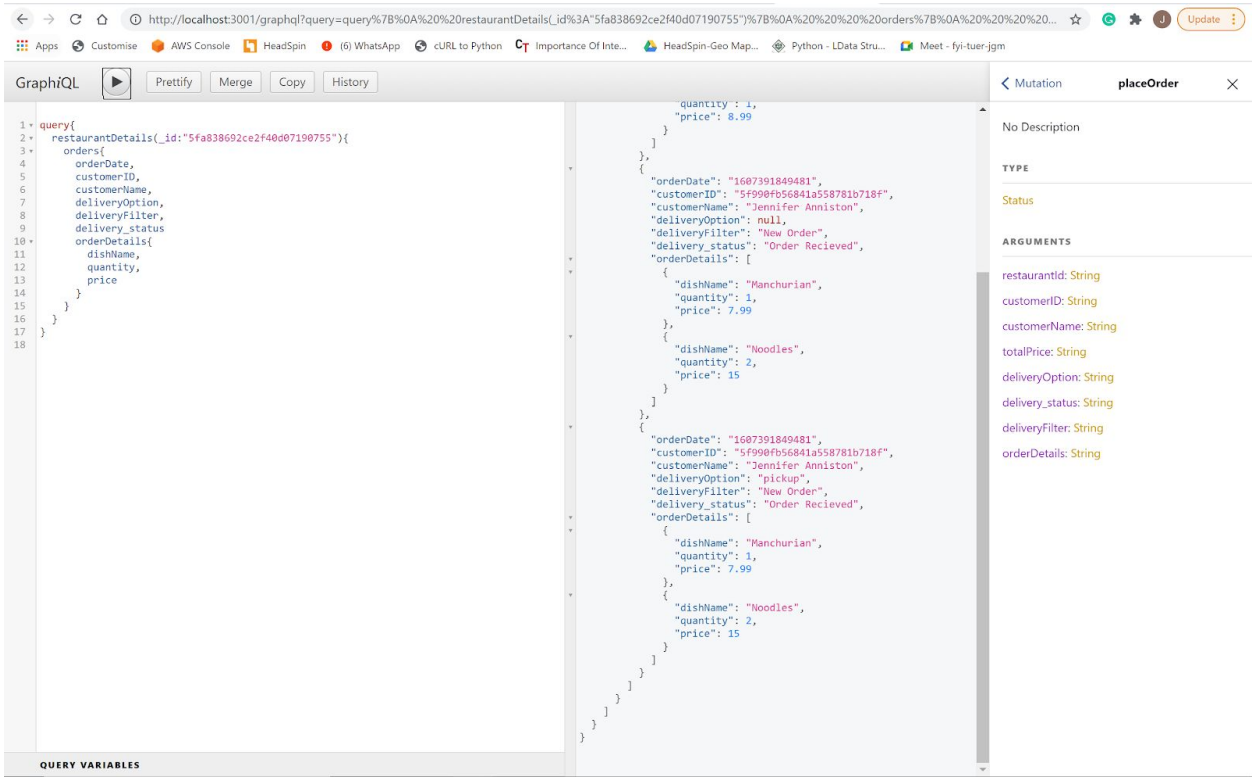
6. Page showing input and output for Restaurant Search



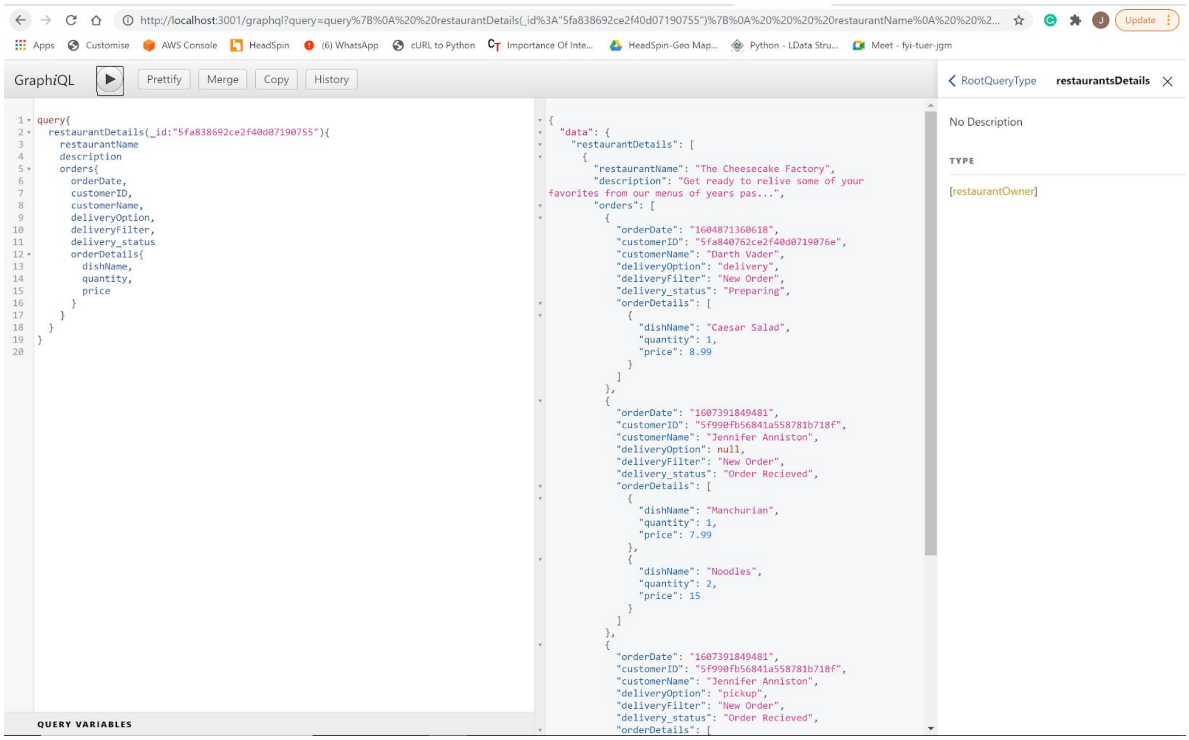
7. Page showing input and output for placing an order.



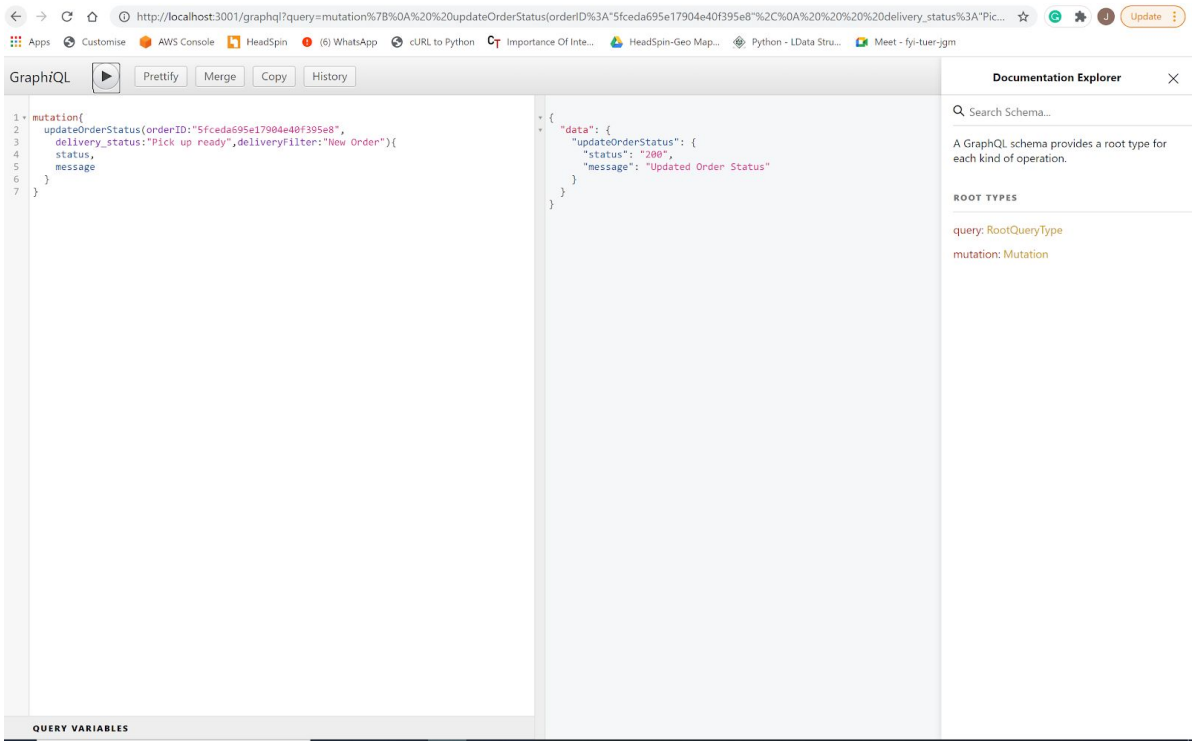
Output:



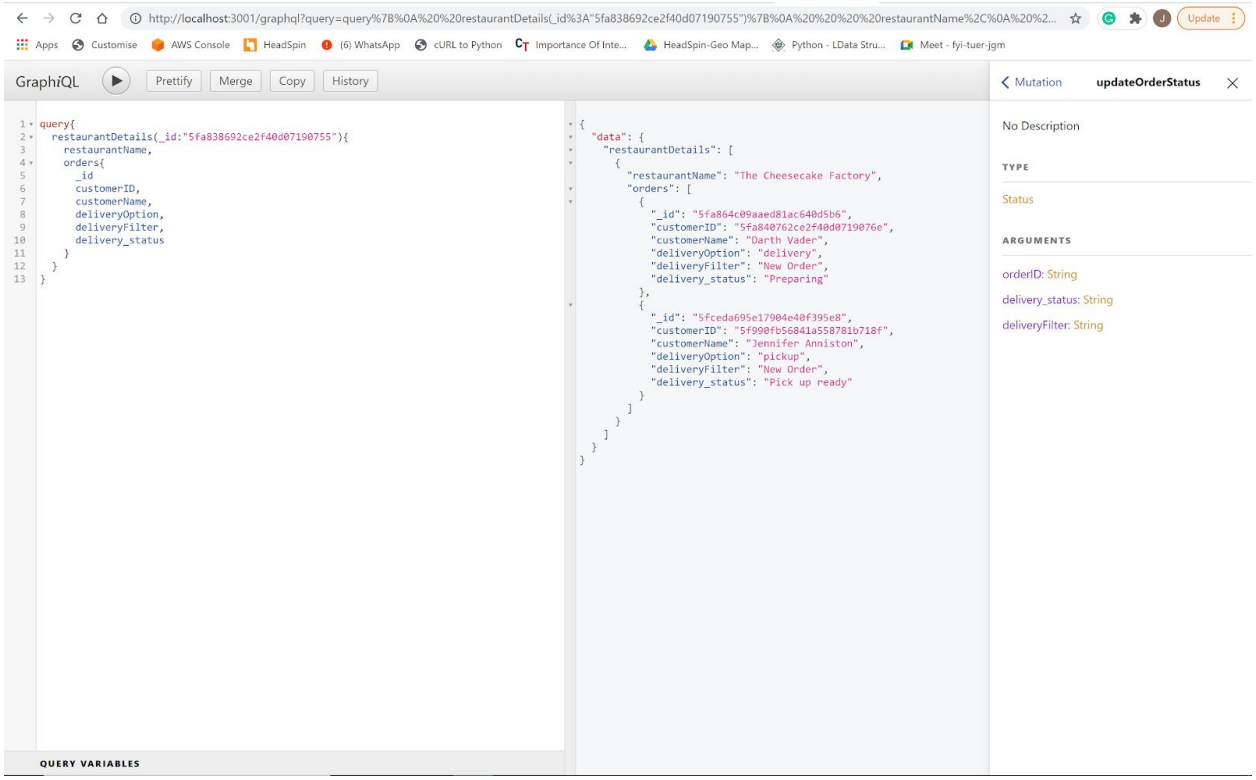
8. Page showing input and output for getting a list of orders for a restaurant



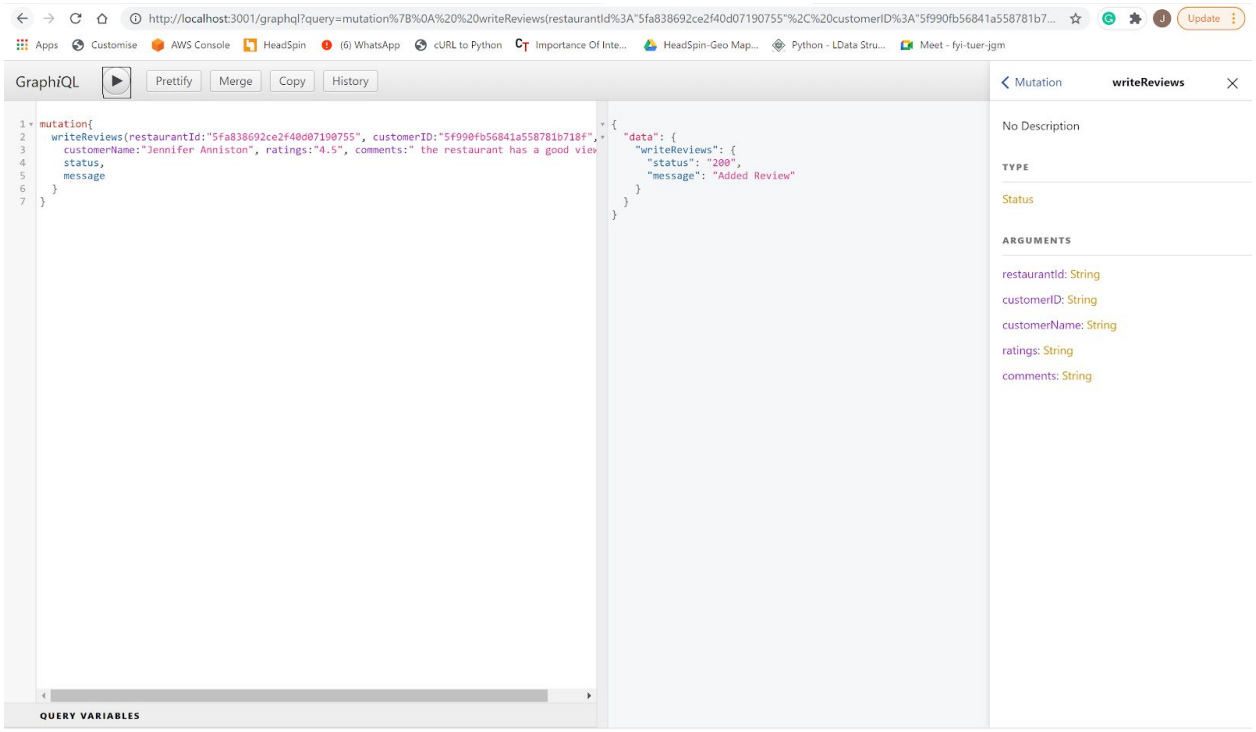
9. Page showing input and output for updating delivery status for an order.



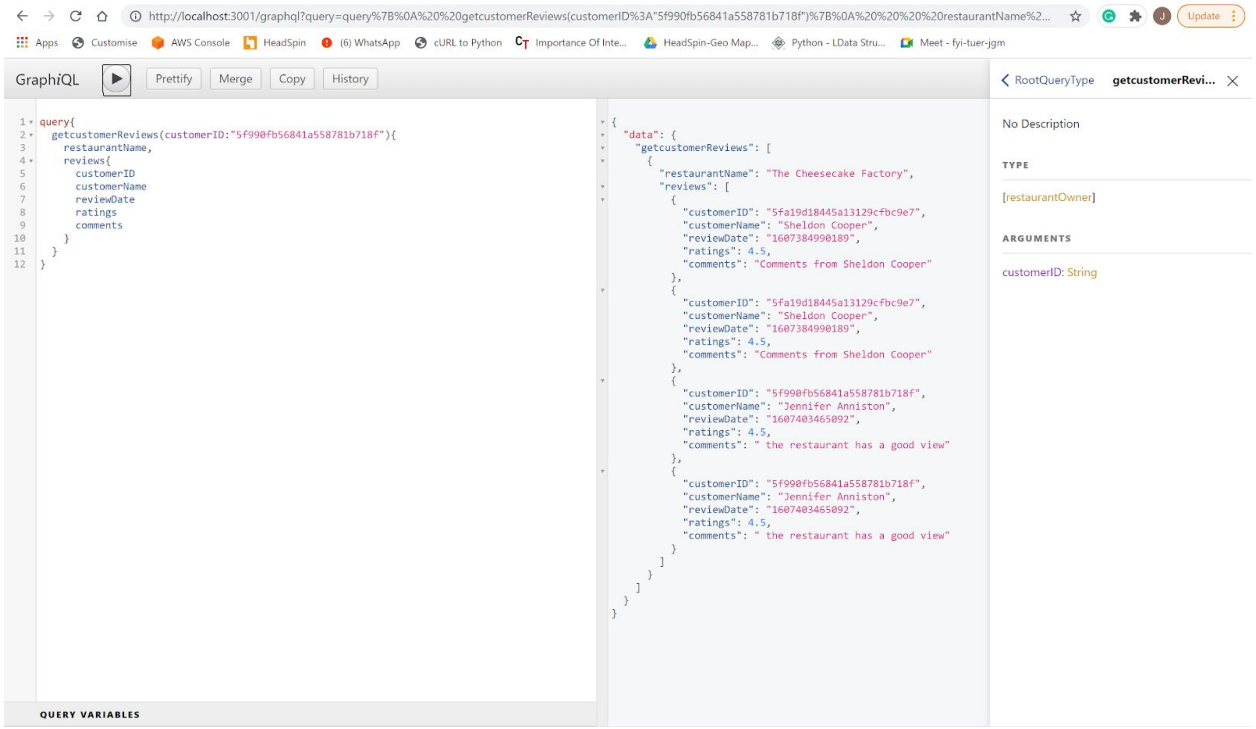
Output:



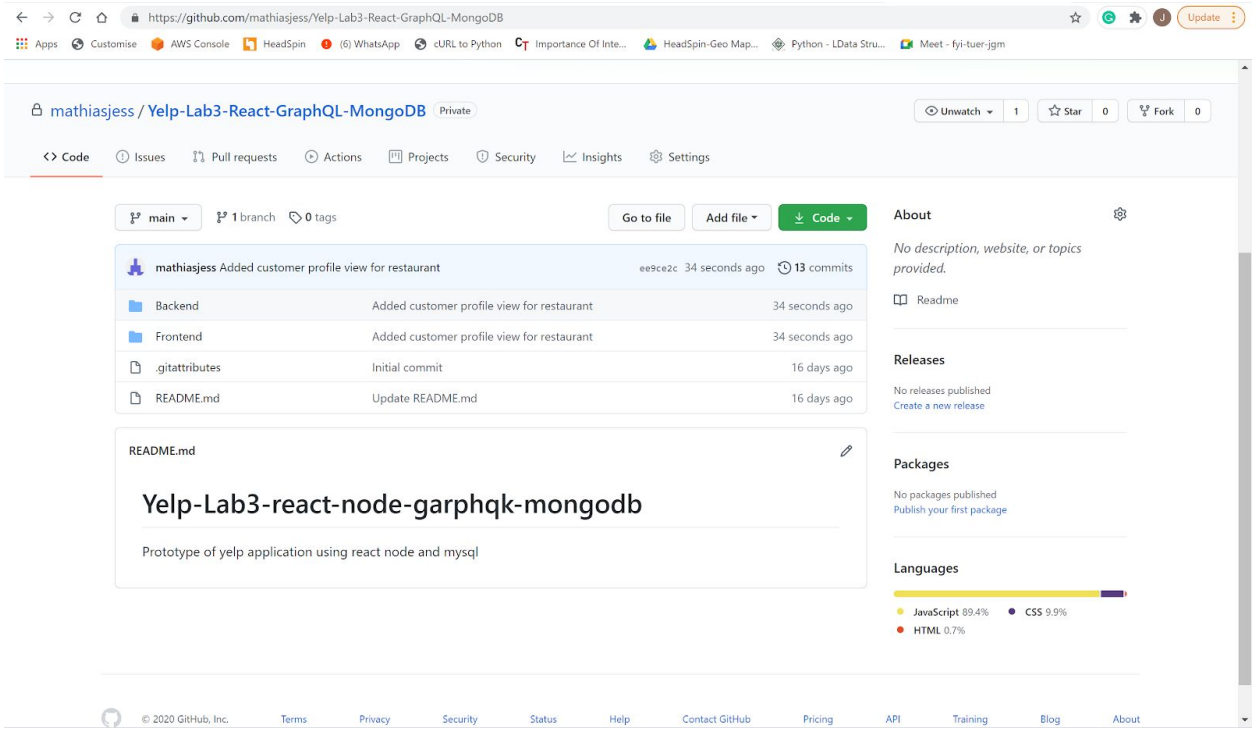
10. Write a customer review

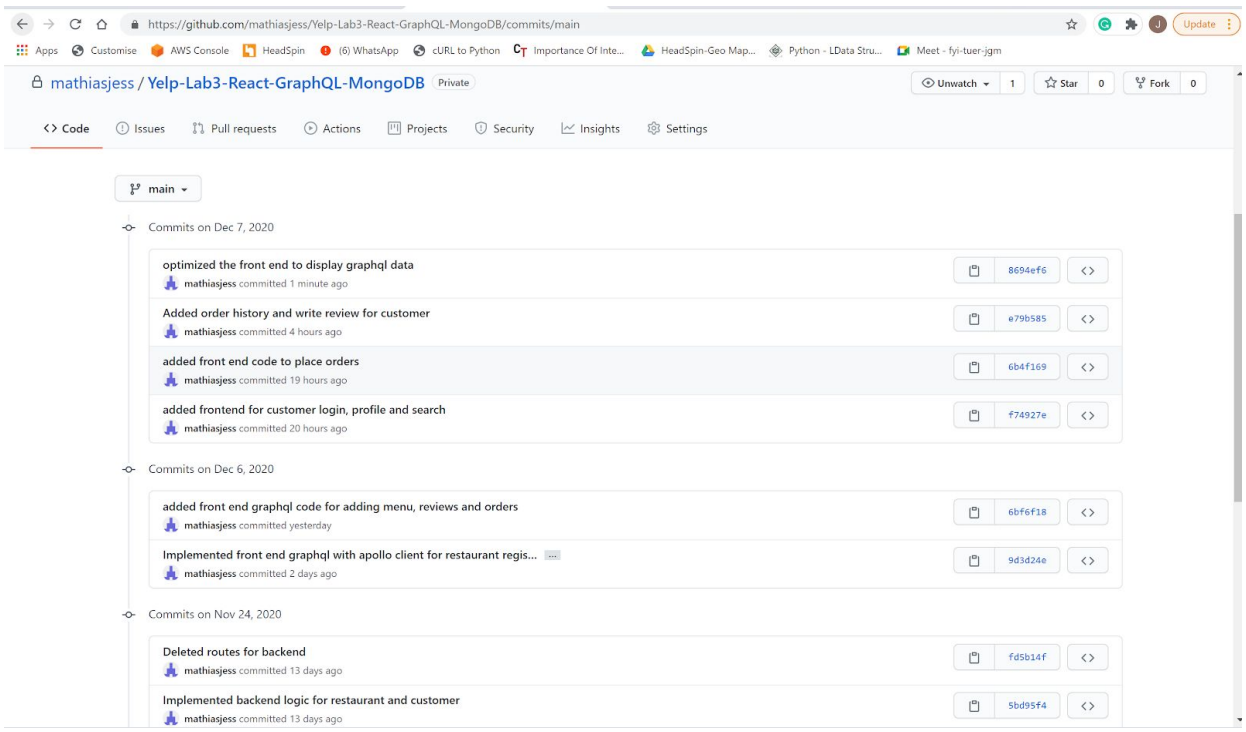


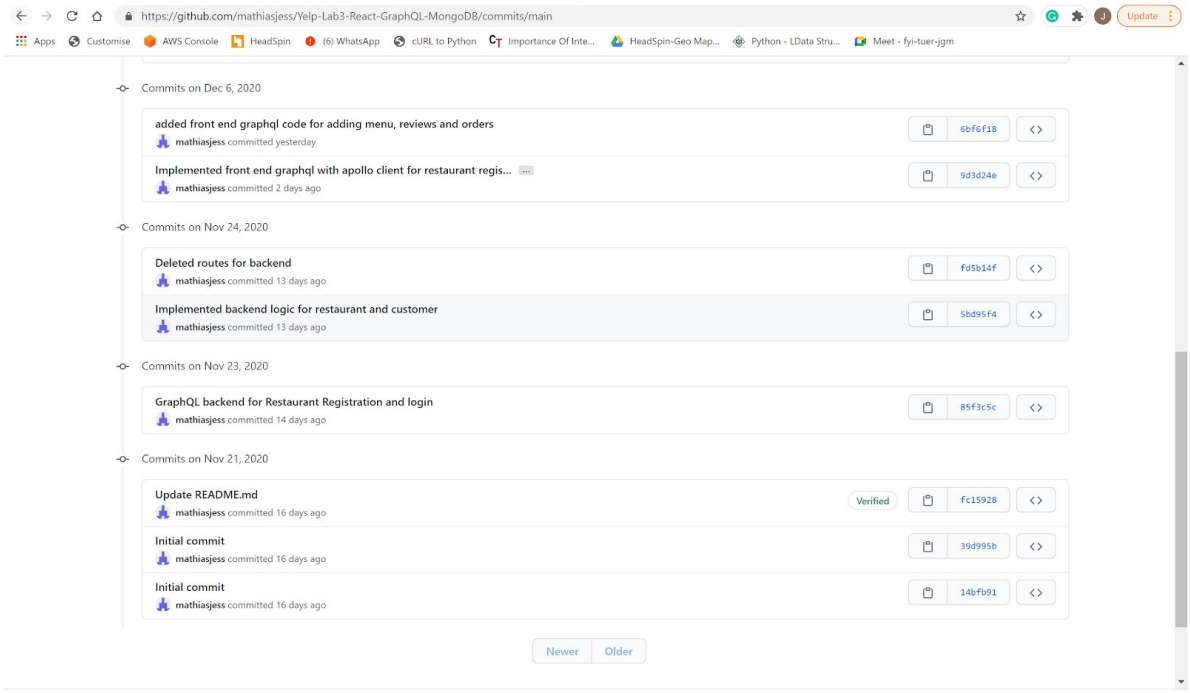
11.Get customer reviews

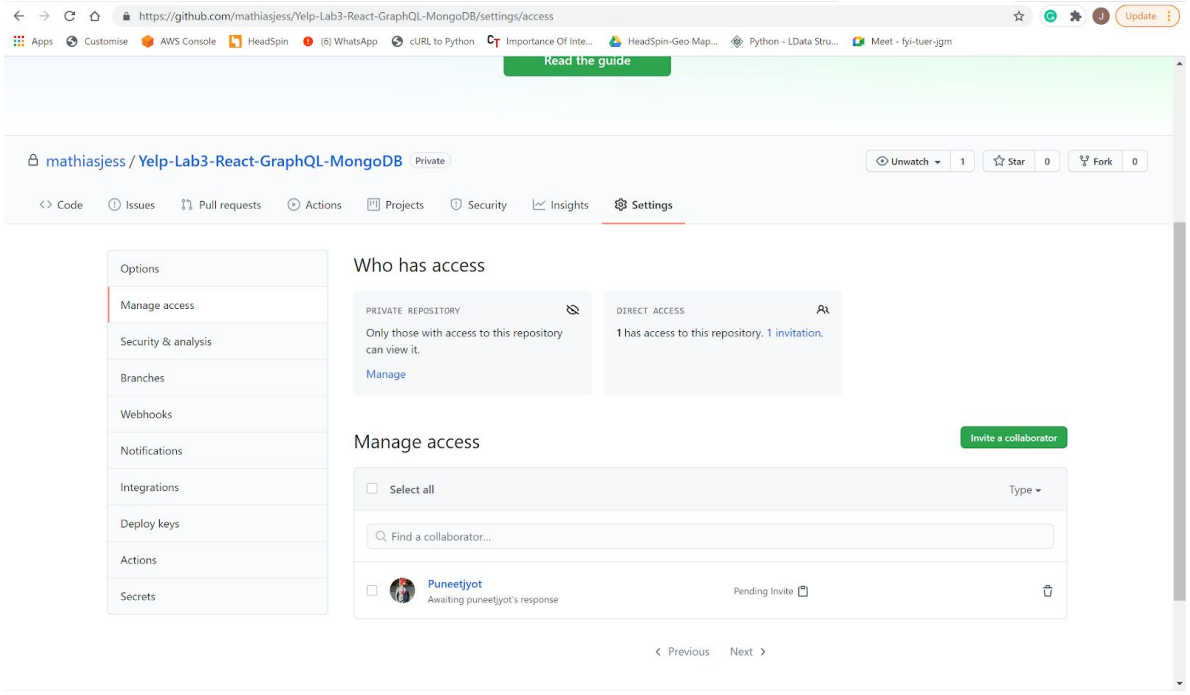


7. **Screenshots of Github Repo**






https://github.com/mathiasjess/Yelp-Lab3-React-GraphQL-MongoDB/commits/main
Commits on Dec 6, 2020
added front end graphql code for adding menu, reviews and orders
mathiasjess committed yesterday
6bf6f18
Implemented front end graphql with apollo client for restaurant regis...
mathiasjess committed 2 days ago
9d3d24e
Commits on Nov 24, 2020
Deleted routes for backend
mathiasjess committed 13 days ago
fd5b14f
Implemented backend logic for restaurant and customer
mathiasjess committed 13 days ago
5bd95f4
Commits on Nov 23, 2020
GraphQL backend for Restaurant Registration and login
mathiasjess committed 14 days ago
85f3c5c
Commits on Nov 21, 2020
Update README.md
mathiasjess committed 16 days ago
Verified
fc15928
Initial commit
mathiasjess committed 16 days ago
39d995b
Initial commit
mathiasjess committed 16 days ago
14bfb91
Newer
Older


https://github.com/mathiasjess/Yelp-Lab3-React-GraphQL-MongoDB/settings/access
Read the guide
mathiasjess / Yelp-Lab3-React-GraphQL-MongoDB
Private
Unwatch 1
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
Options
Manage access
Security & analysis
Branches
Webhooks
Notifications
Integrations
Deploy keys
Actions
Secrets
Who has access
PRIVATE REPOSITORY
Only those with access to this repository can view it.
Manage
DIRECT ACCESS
1 has access to this repository. 1 invitation.
Manage access
Invite a collaborator
Select all
Type
Find a collaborator...
Puneetjyot
Awaiting puneetjyot's response
Pending Invite
Previous
Next