Jessica Mathias Lab Report 1 -Yelp-Prototype

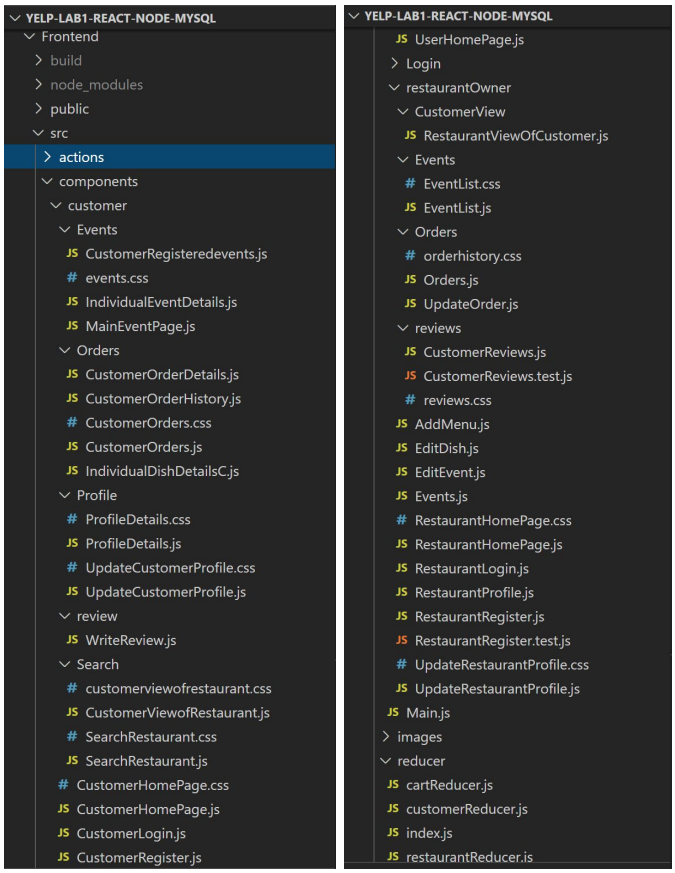
Demo Video Youtube Link: https://www.youtube.com/watch?v=VrpoNWg-bGI

1. Introduction

The objective of lab1 is to build a working prototype of the yelp application. The technologies used for this application are Node JS in the backend, react in the front end, mysql as the database and AWS EC2 instance to deploy the application. This lab focuses on implementing restful services to create, edit, update and delete data.

2. System Design

FrontEnd: The yelp application is a food and lifestyle website where users can search for restaurants, dishes, events etc. This app also serves as a website where users can order food. The application is designed for two end users namely the customer and the restaurant owner. While the customer front end allows the end user to search for restaurants, events, order food and register for events, the restaurant owner is required to have additional features where he/she can update the order status and view the customers who have registered for the events. In other words, the front end is expected to optimise for dynamic real time changes in data



Usage of API's for the role of customer and Restaurant Owner

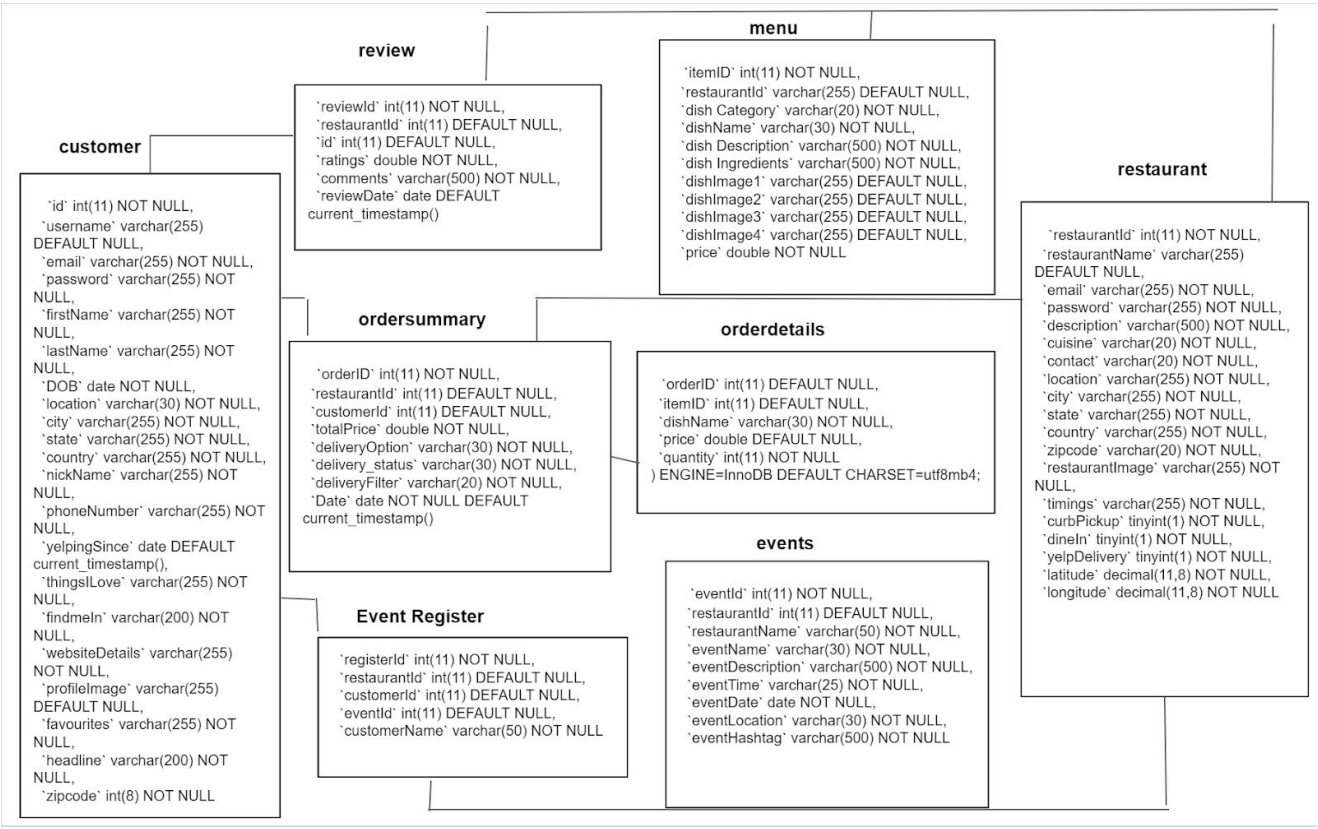
#	API	Method	Usage
1.	'/restaurantregister' '/customerresister', '/restaurantlogin', '/customerlogin, '/updateMenu', '/editMenu', '/addevent'	POST	To register restaurantOwner and customer details and log them in if registered in the database, To add menu details and event details
2.	'/restaurantprofileUpdate/:id' '/updateorderstatus','/cancelorder'	PUT	To update restaurantOwner details, and order details
3.	'/restaurantprofiledetails/:id', '/fetchMenu/:id', '/fetchdish/:id', '/customerprofile', '/getcustomerreview','/fetchregistry/:id', '/restaurantordersummary/:id', '/fetchrestaurantorderdetails/:id', '/individualrestaurantordersummary/:id', '/getrestaurantreview'	GET	Fetching the profile data, menu data, individual dish details, order details and event registry details
4.	'/deleteMenu/:id', '/deleteEvent/:id'	DELETE	Delete a dish, delete an event
5.	'/updatecustomerprofile'	PUT	Updates profile Image and basic details of customer
6.	'/writereview', '/sendorderdetails/:id', '/sendordersummary',	POST	Inserts reviews, order details and order summary
7.	'/getrestaurantreview/:id', '/getcustomerreview/:id', '/fetchsinglevent/:id', '/fetchcustomerorderdetails/:id', '/fetchcustomerordersummary/:id', '/searchforrestaurant'	GET	Fetches reviews, order details and event details for the customer
8.	'/deleteregisteredevent/:id'	DELETE	User deletes a registered event

MYSQL Database:
The two main tables in the yelp database are customer and restaurant

1. Customer: The customer table consists of basic information of the user such as location, interests, favourites, data of birth, profile picture etc
2. Restaurant: The restaurant table consists of details of the restaurant such as name, description, location, contact, mode of delivery etc
3. Order Summary: The order summary consists of the restaurant Id, customerId, total value of order, order status and type of order
4. Orderdetails: the order details consist of individual items in an order along with their quantity and price
5. Events: This table consists of event details such as event name, description, date, timings and location
6. Eventregistry: This table consists of customerIDs who have registered for various events
7. Menu: This table consists of each dish for all restaurants
8. Review: The reviews written by a customer are stored and retrieved from this table

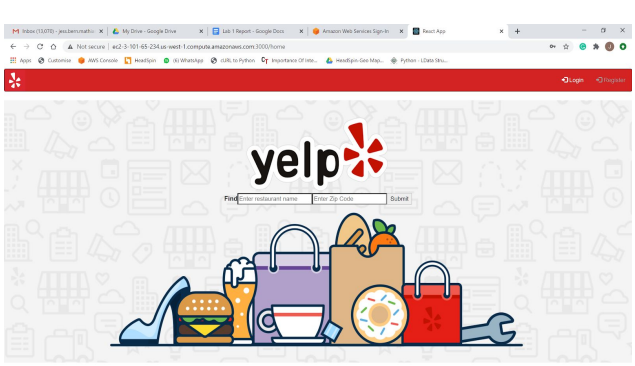
Schema Diagram

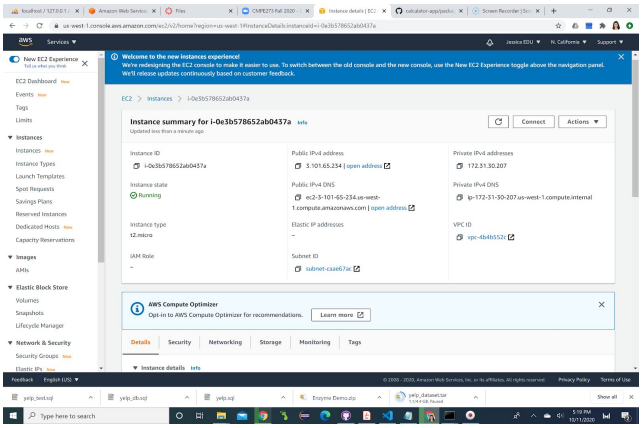
Yelp Database



Deployment:

1. The yelp application was deployed on amazon ecs http://ec2-3-101-65-234.us-west-1.compute.amazonaws.com:3000/
2. Mysql was deployed on Amazon RDS and accessed locally through mysql workbench

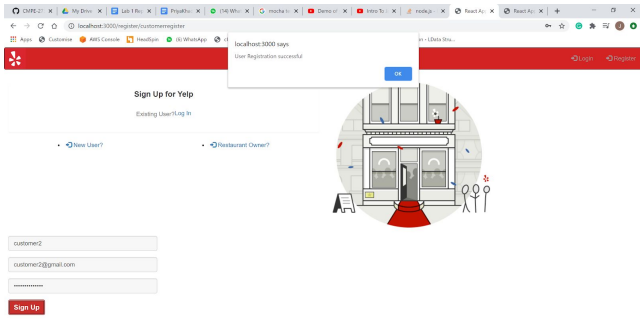




3. **Results:**

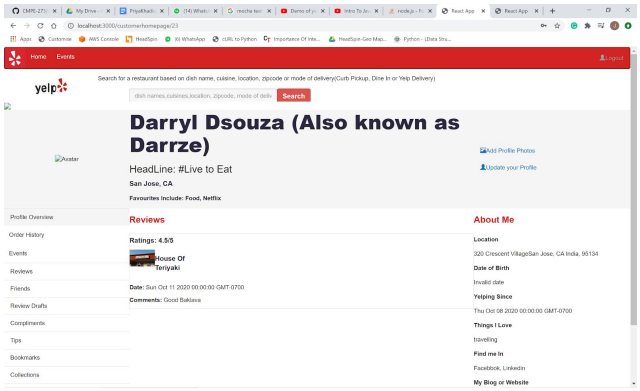
Customer Register:

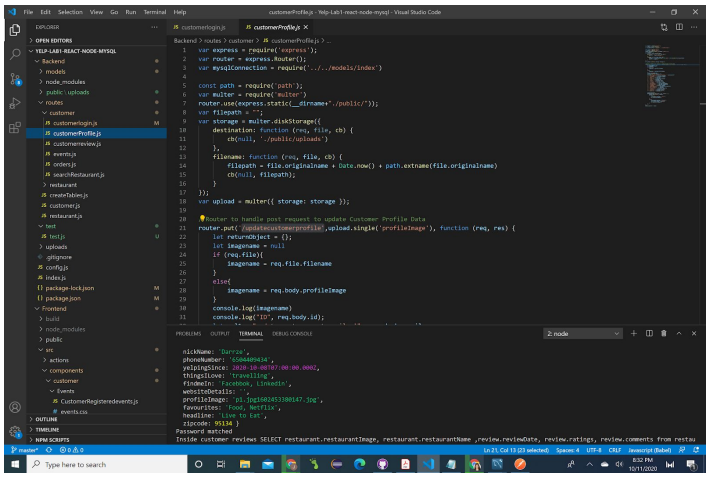
Customer Registers using the name, email and password, on successful registration, he is redirected to existing user log in



Existing user Log In:

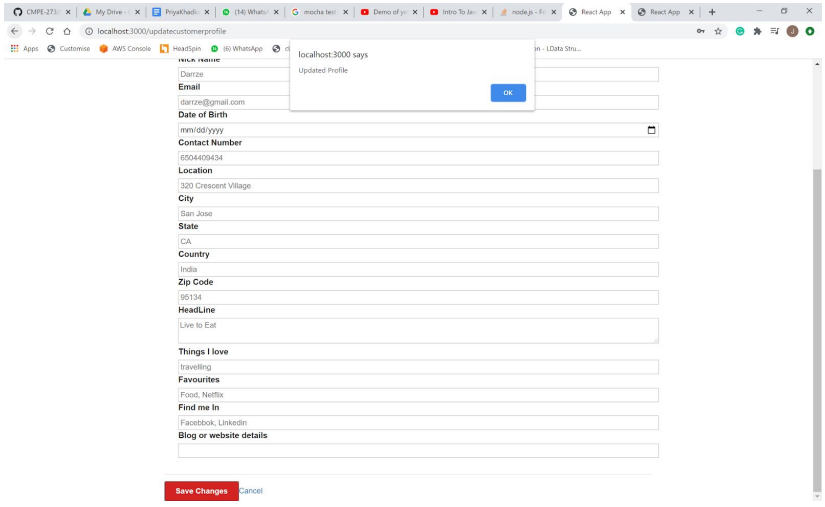
Once the user has logged in, he will be able to see his profile

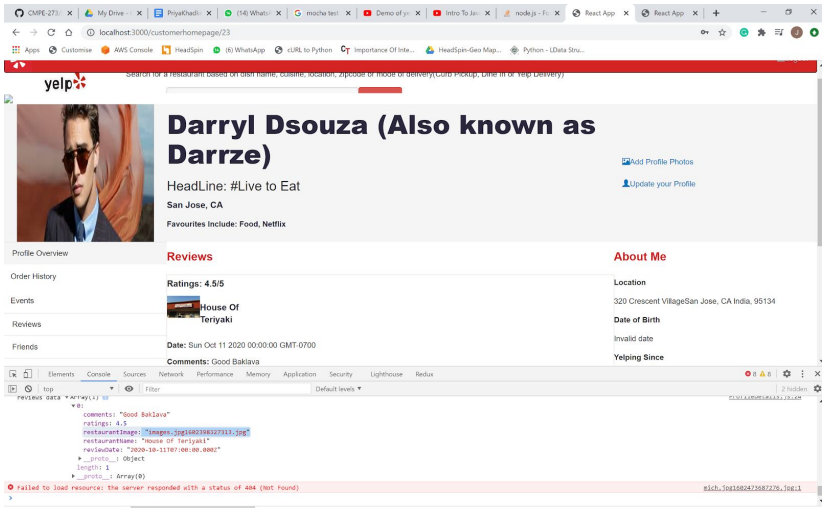




Updating the profile:

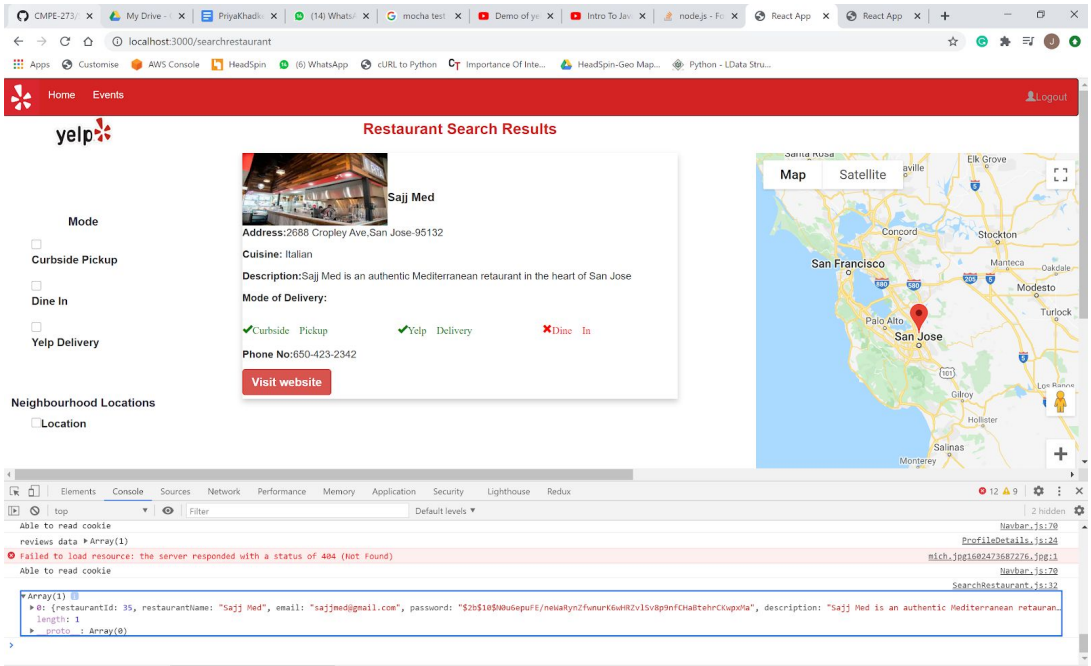
A user can update his basic details including profile photo. The updated details should reflect on the profile





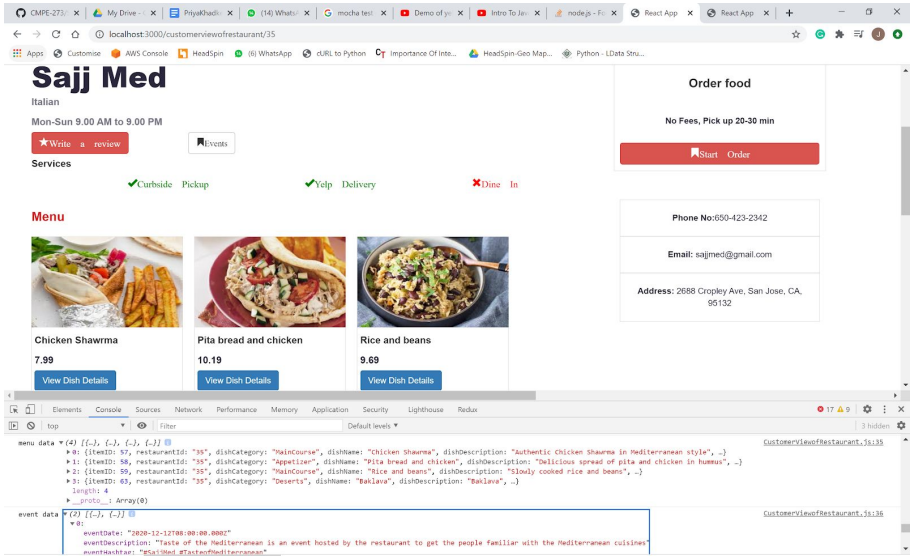
Searching for a restaurant:

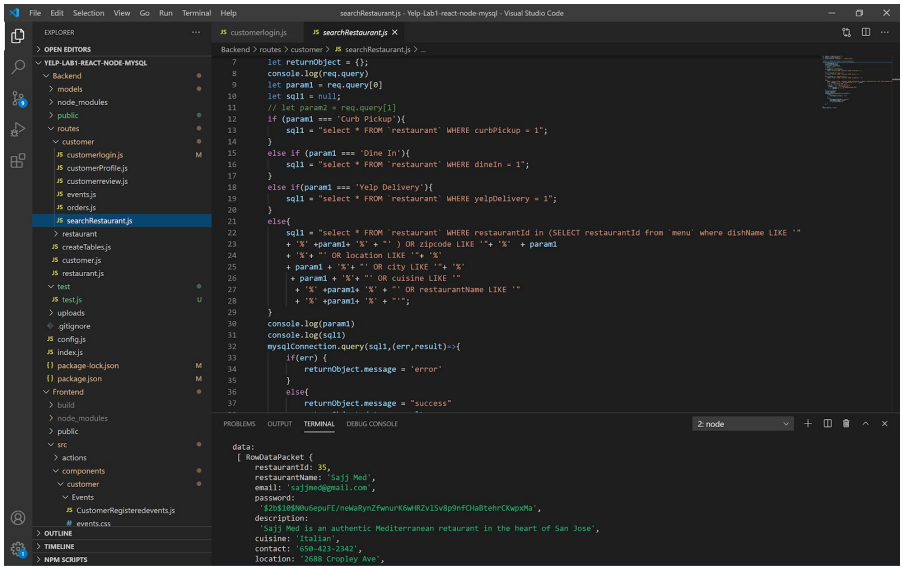
The user should be able to search for a restaurant using name, cuisine, dish, location and zipcode. As seen in the screenshot below, the maps shows the location for the restaurant. The user is further able to filter his search based on pickup, delivery or zipcode



View Restaurant:

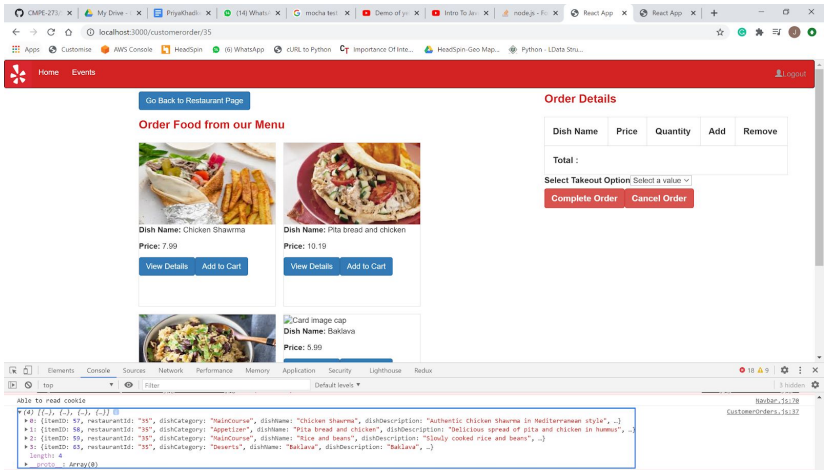
On clicking the restaurant website, the user should be able to see the restaurant page with restaurant details, dishes, events and reviews

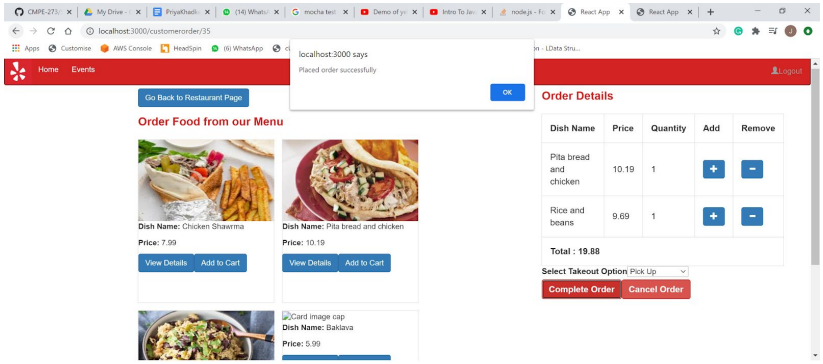


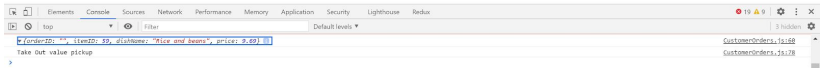


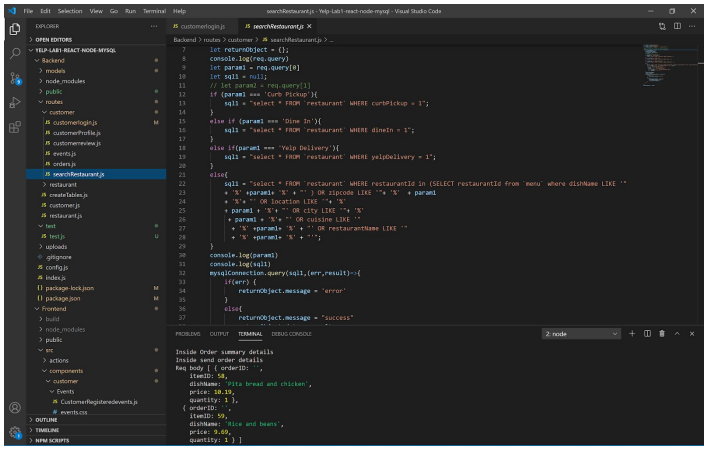
Order Food:

The user should be able to order food from the restaurant menu. Based on the option he selects for takeout(namely pickup and delivery) he will be able to see the order history.



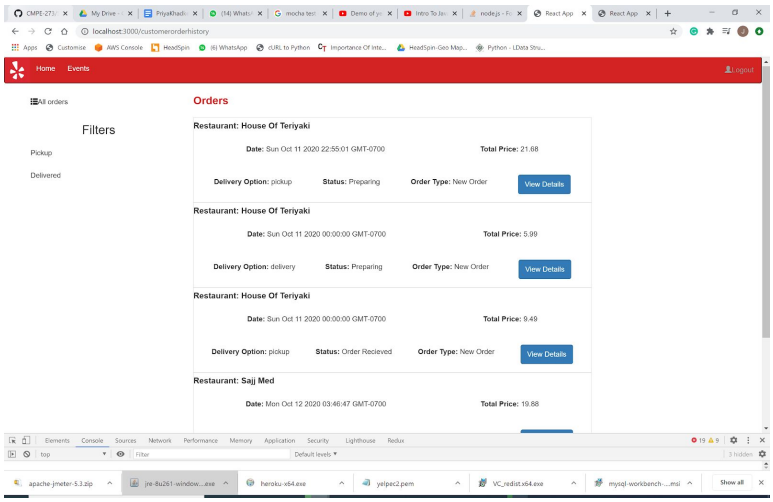


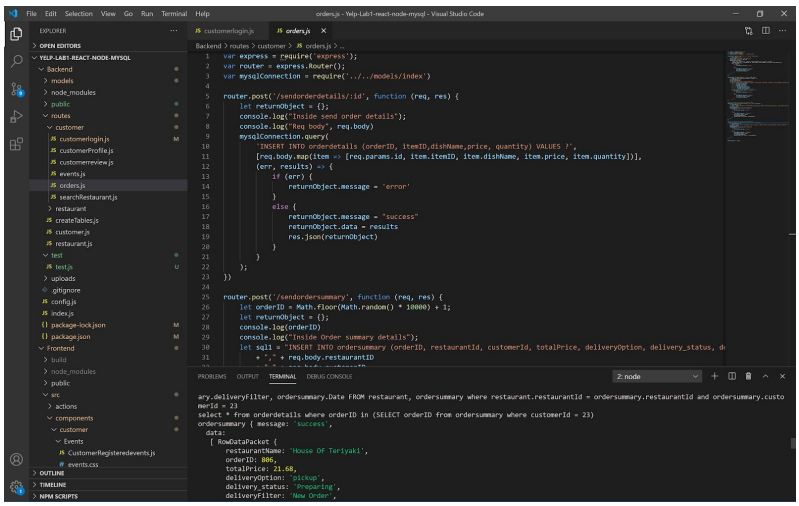




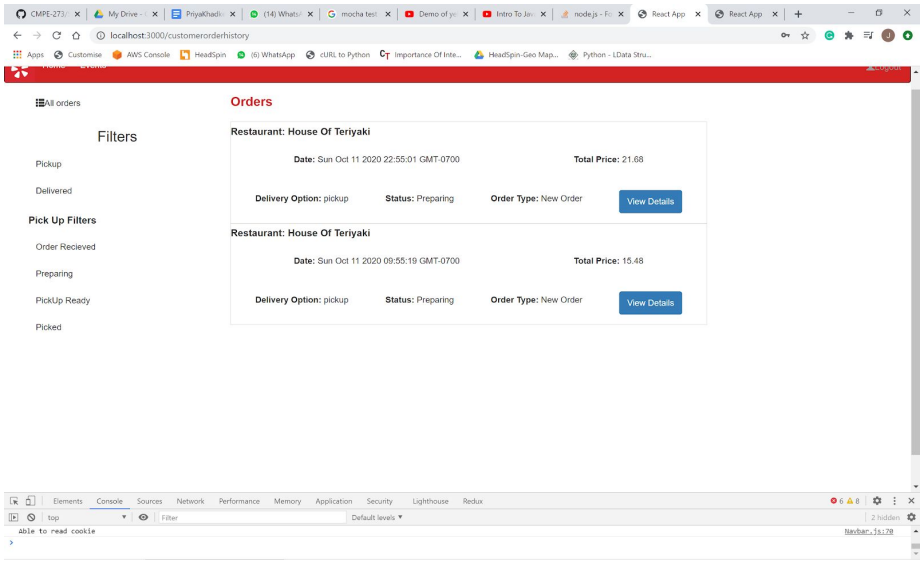
Order History:

Once the user has ordered, the application is redirected to the order history page. The user will be able to filter his orders based on pickup, delivery, and in each category he will be able to further filter down based on order received, preparing, pick up ready or on the way and picked up or delivered



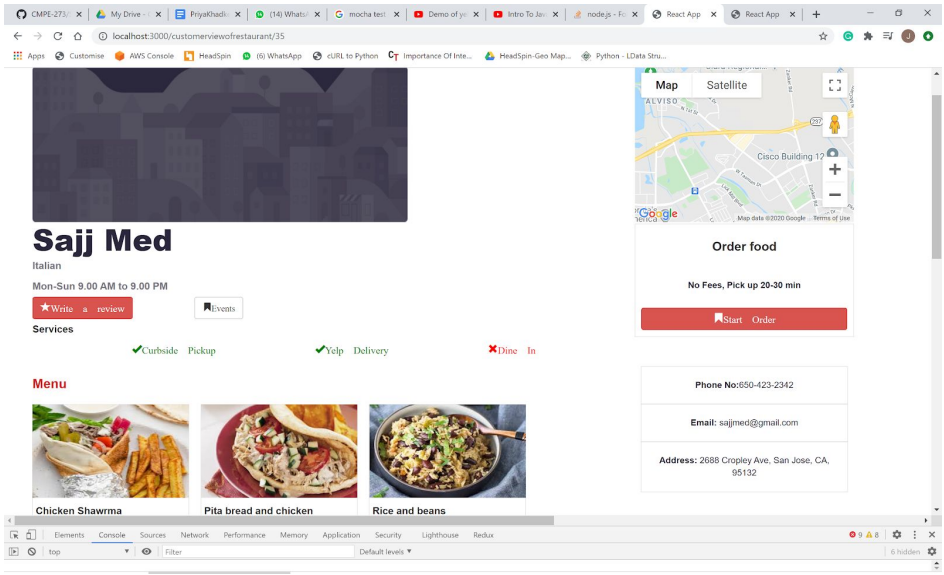


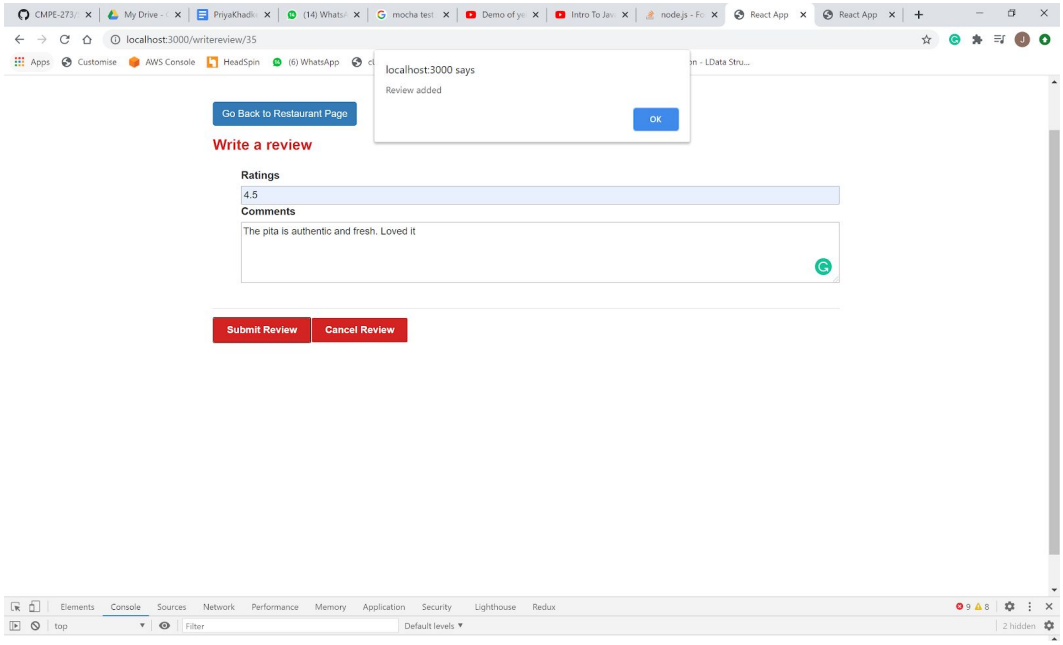
Filter based on order type pickup and order status: Order received



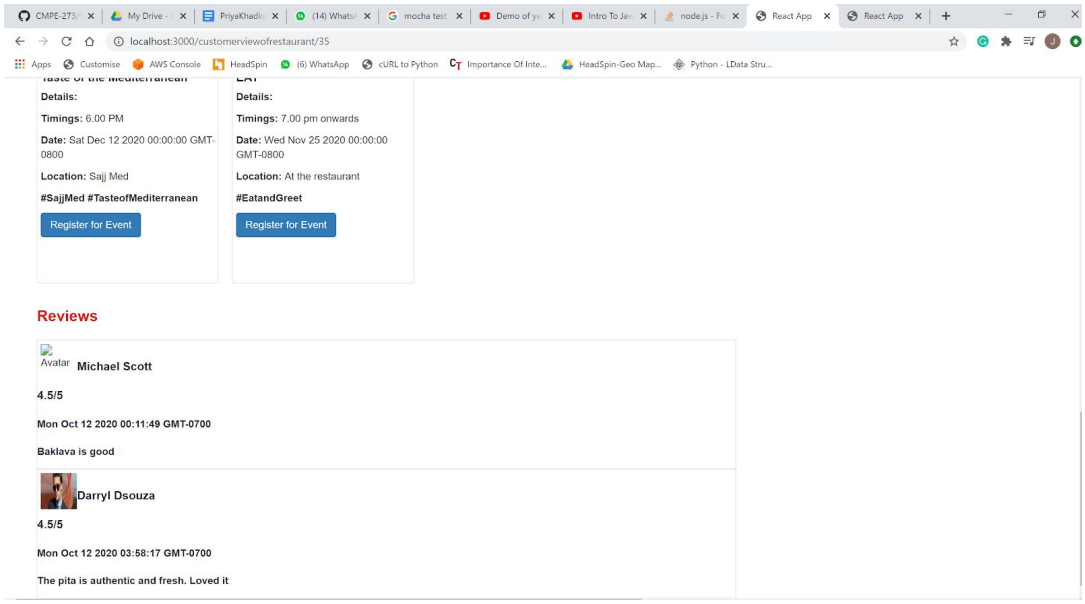
Write a review:

The user should be able to search for a restaurant and write a review. The review should be reflected on the restaurant page



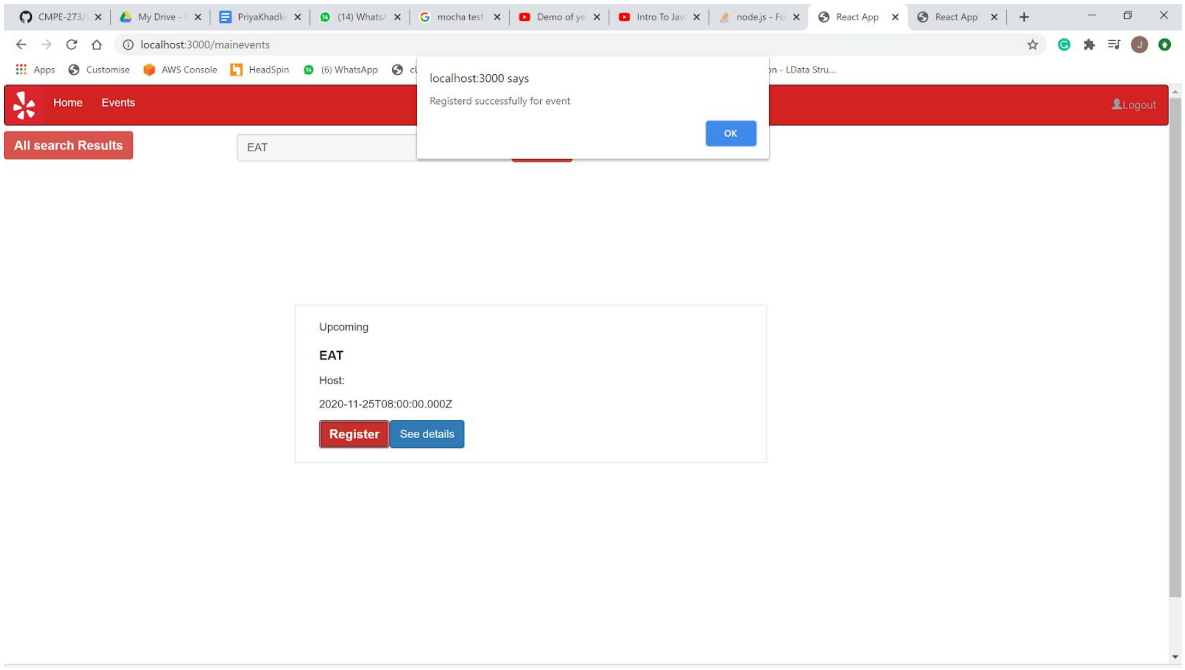


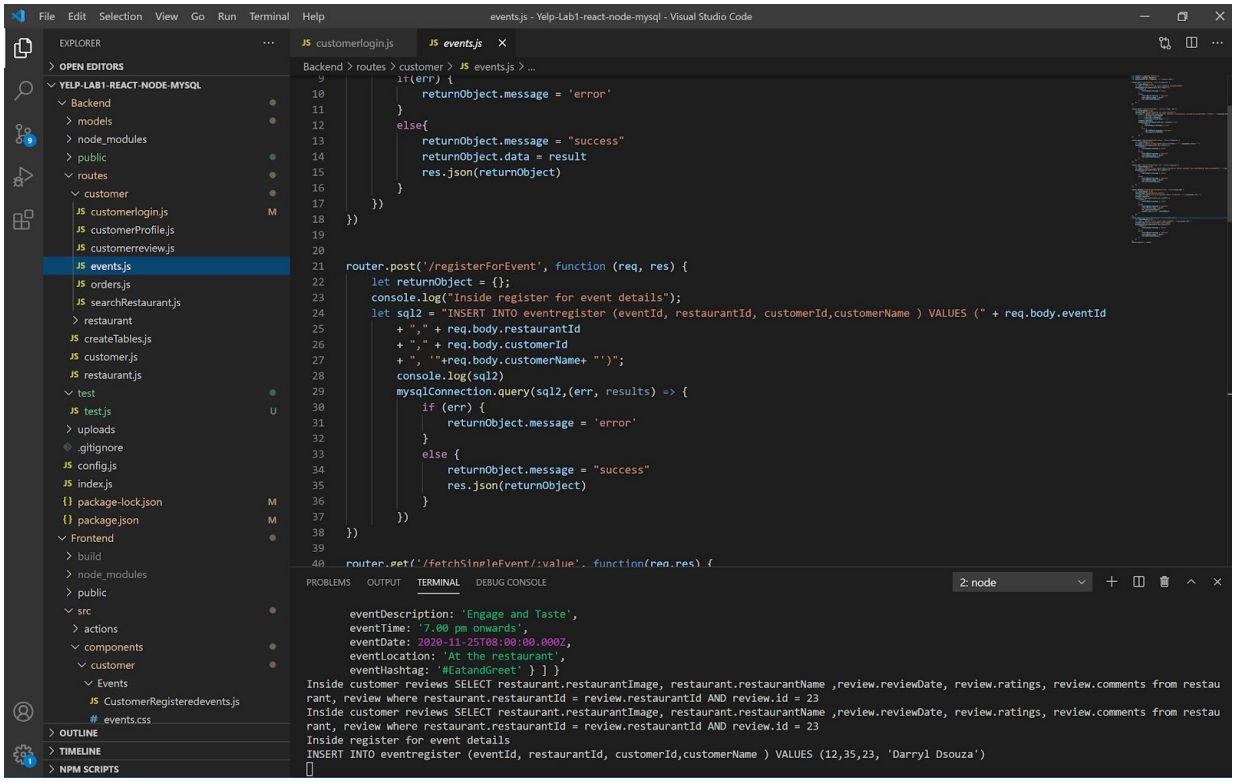
The review is reflected along with data, ratings and comments



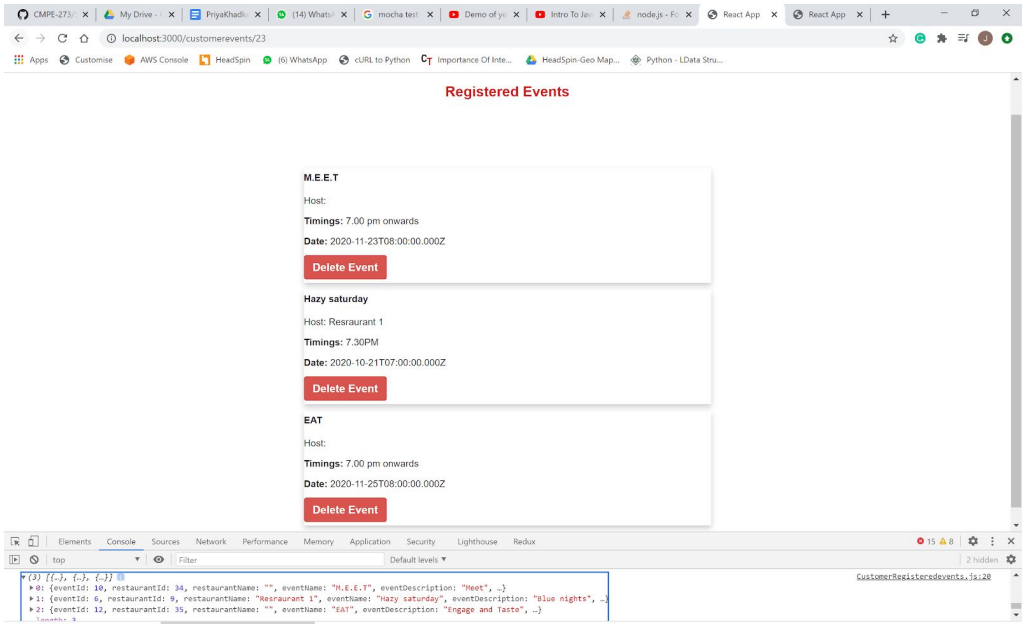
Register for an event:

The user should be able to go the events page, search for an event and register for an event. He should then be able to see registered events





Registered Events:

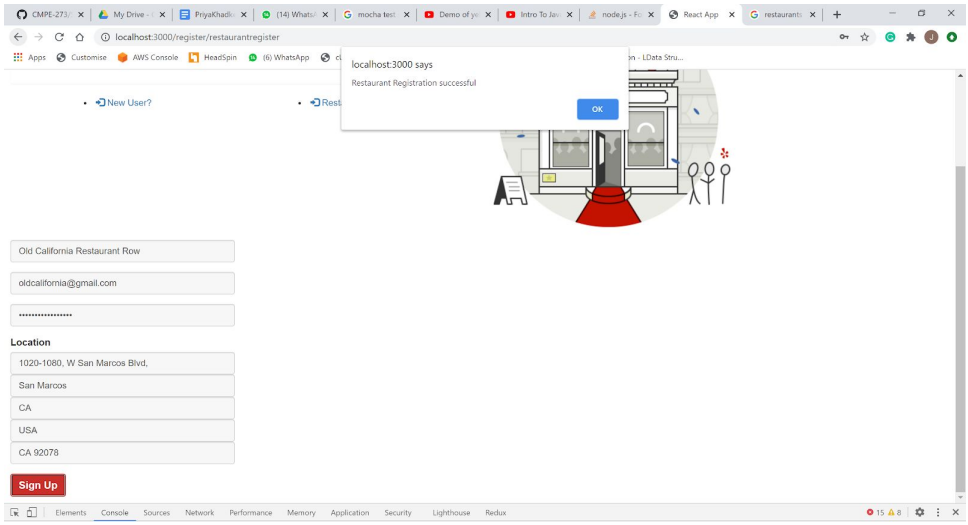


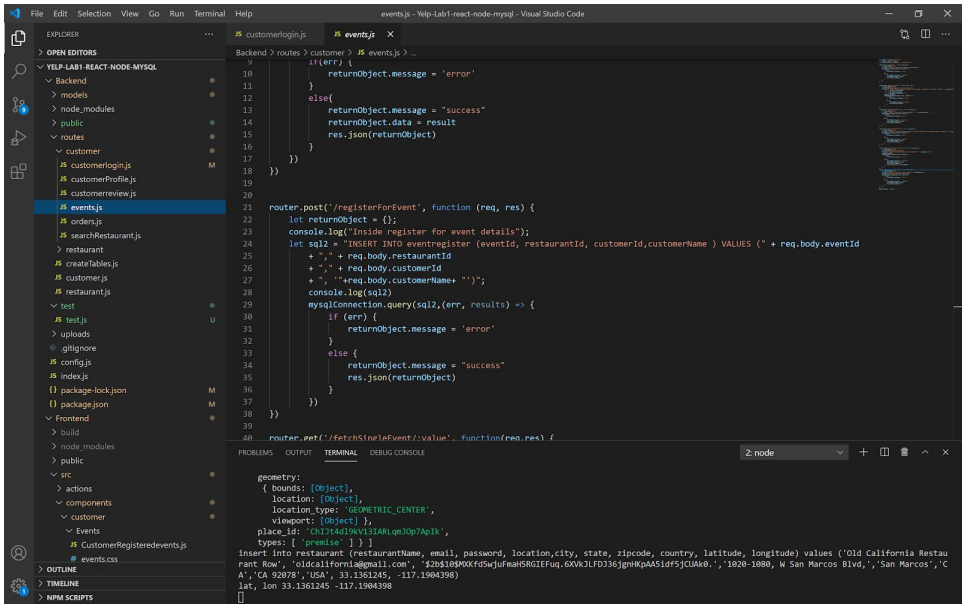
Restaurant Owner

1) **Restaurant owner registration and login:**

When the restaurant owner registers, they need to give their restaurant name and location. Based on the location, the backend will fetch the latitude and longitude and add it to the database.

Whenever the customer searches for a restaurant, he will be able to see the restaurants on the maps because of this

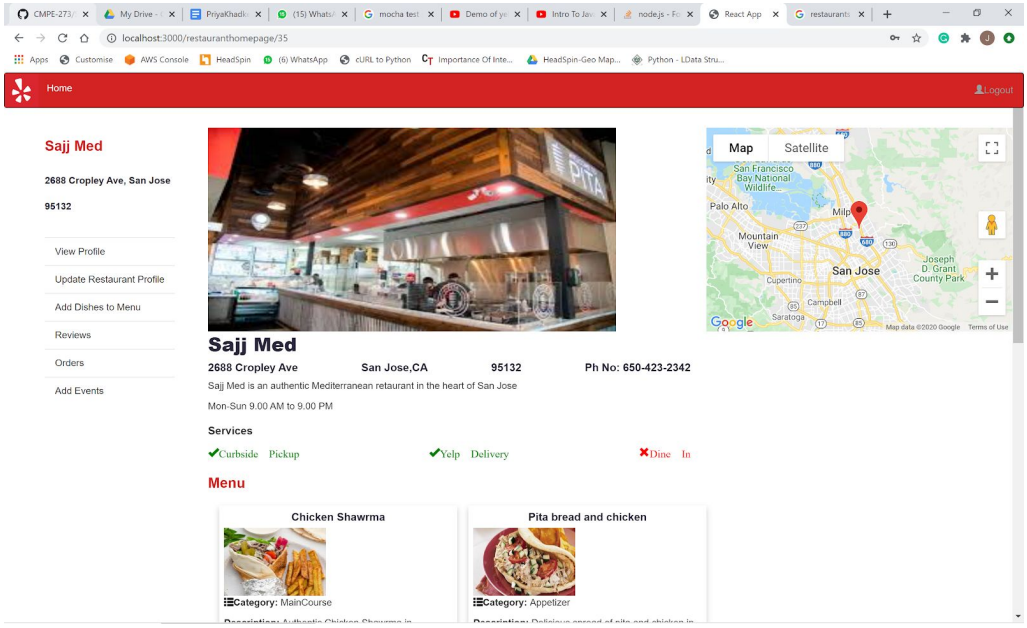


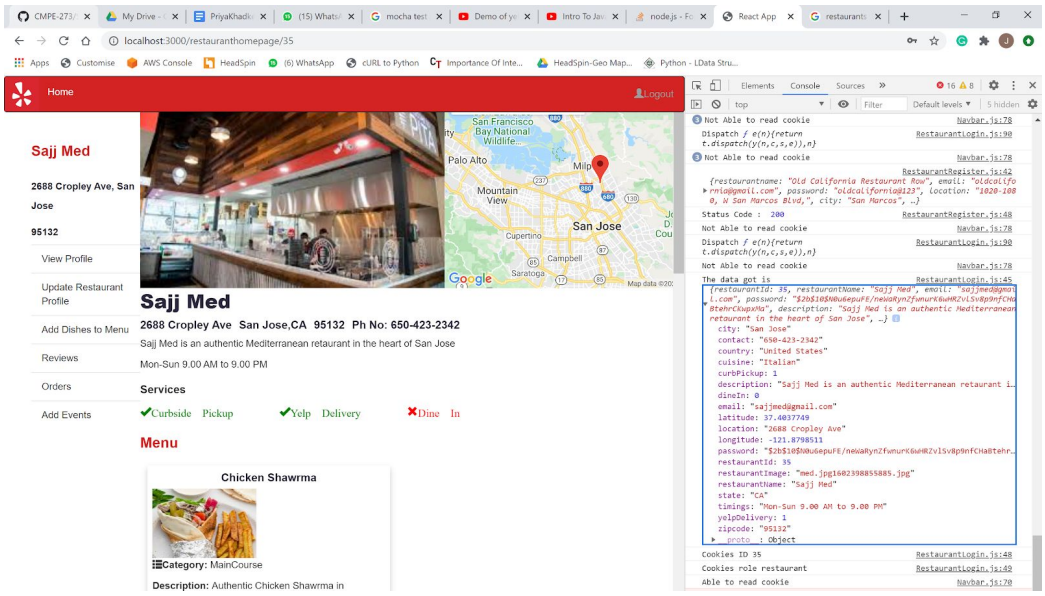


Once the user has logged in, he can view his profile page

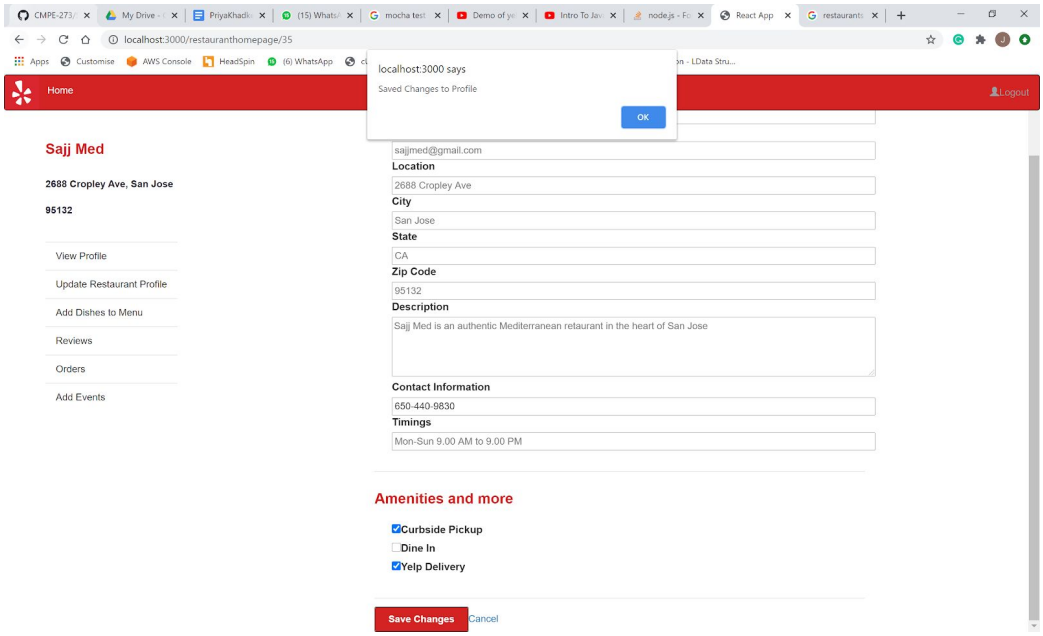
2) Profile Details and Updating the profile

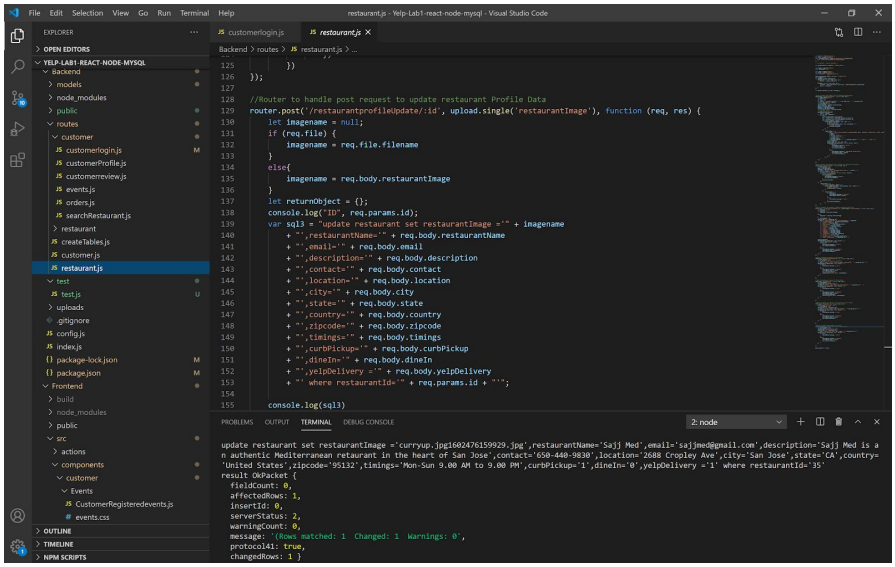
The restaurant owner should be able to see all the profile details including the dishes, events and basic restaurant details.



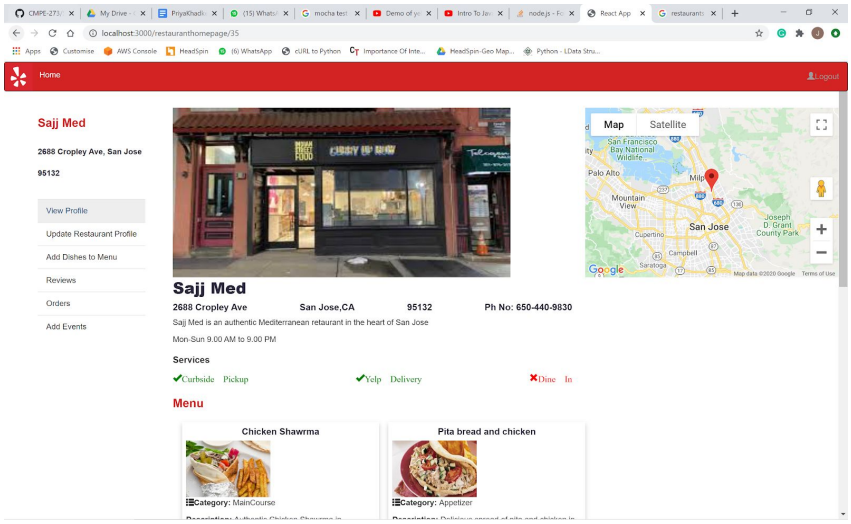


3) Update Profile Image and details



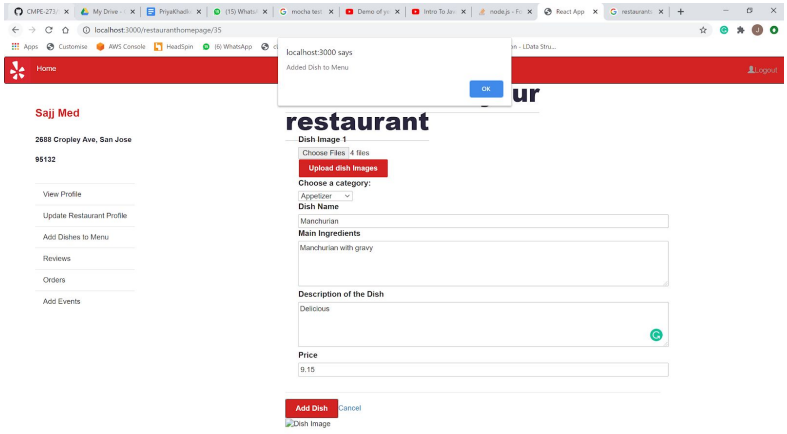


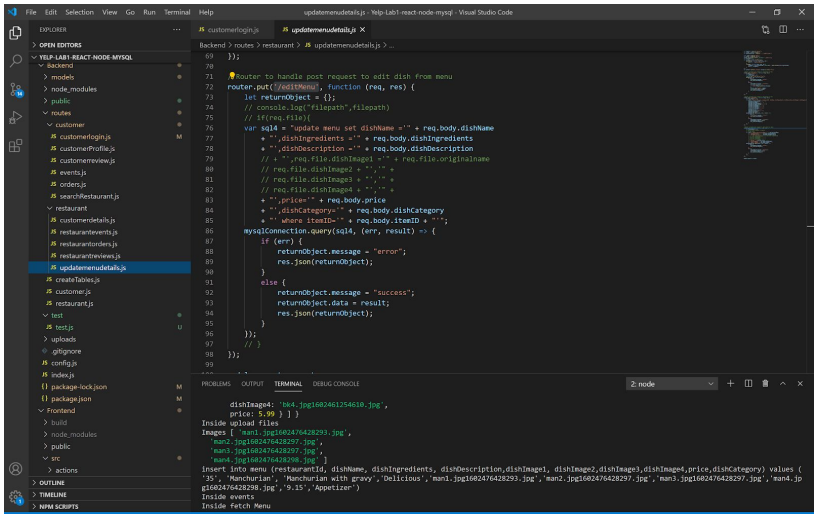
Here we see that the restaurant details have been successfully updated



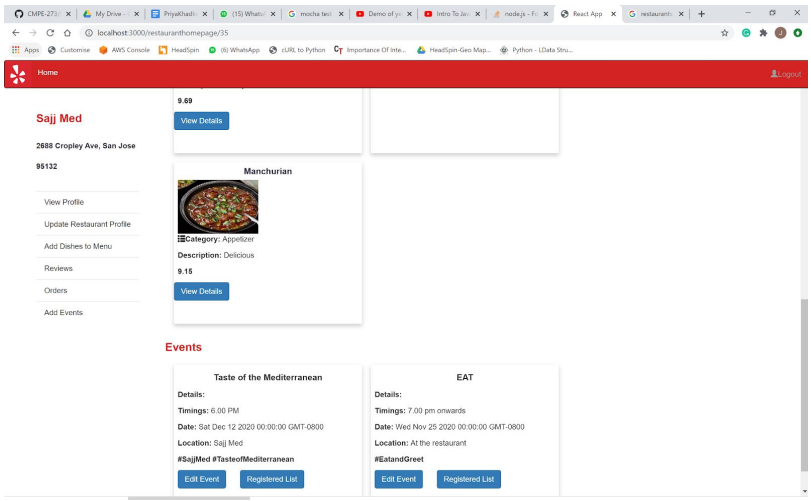
4) **Add dishes to menu**

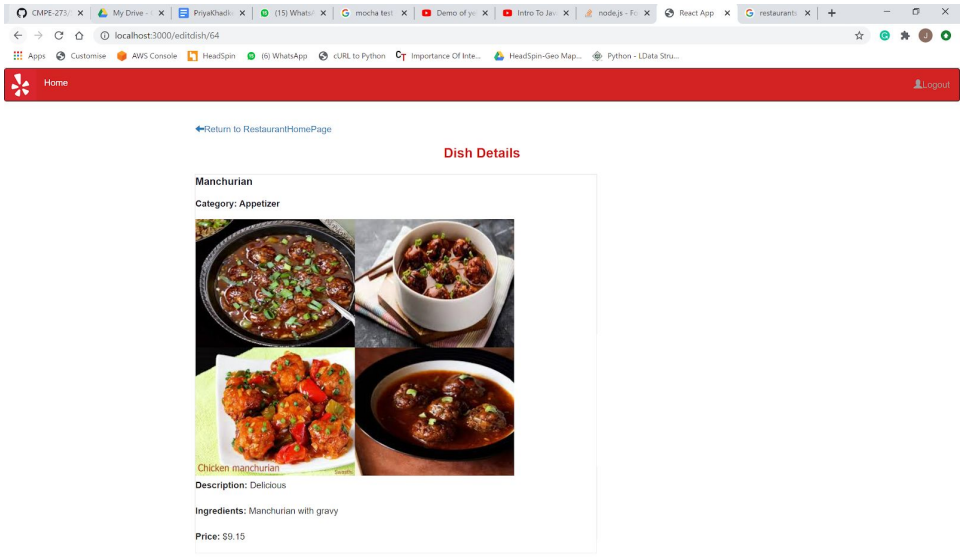
The restaurant owner should be able to add a dish by category with multiple images for each dish





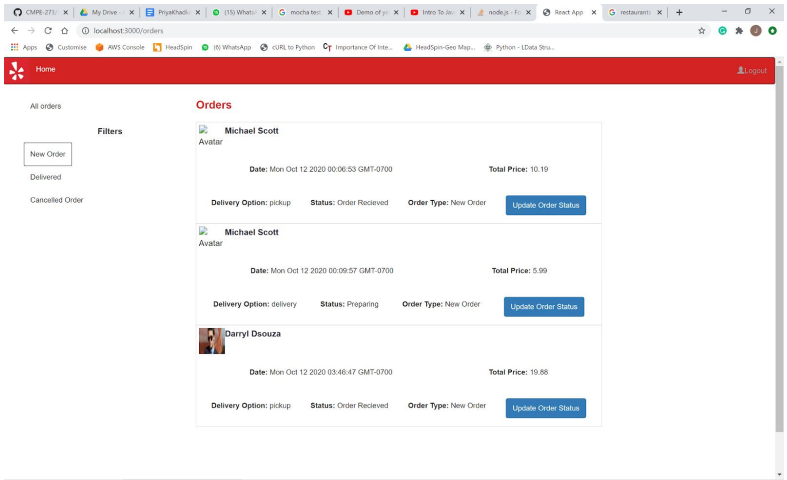
The added dish is reflected in the restaurant profile



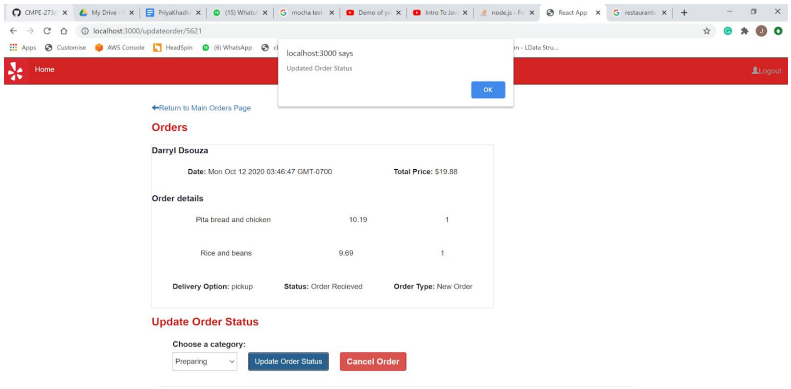


5)Orders:

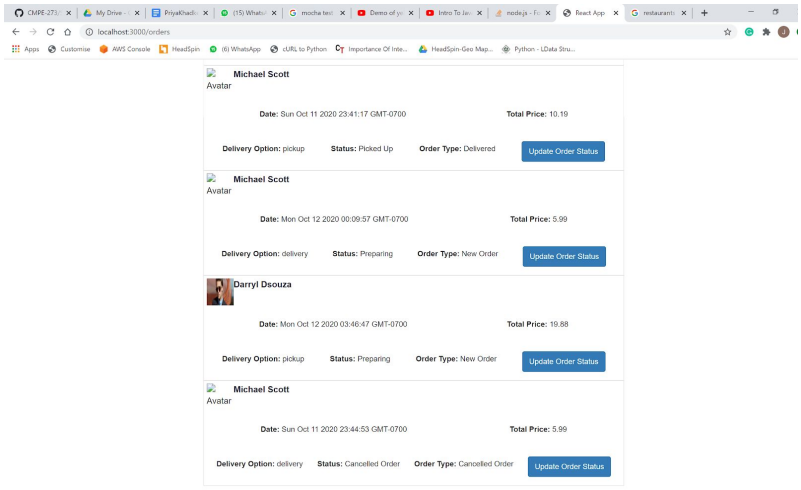
The restaurant owner should be able to view all orders made to his restaurant and filter them based on order status



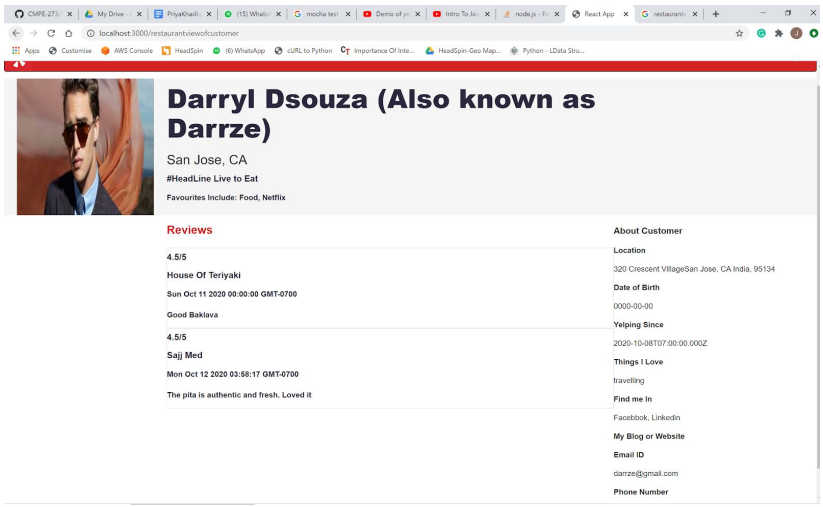
Update Order Status:



The status of Darryl's order is updated to prepaaring

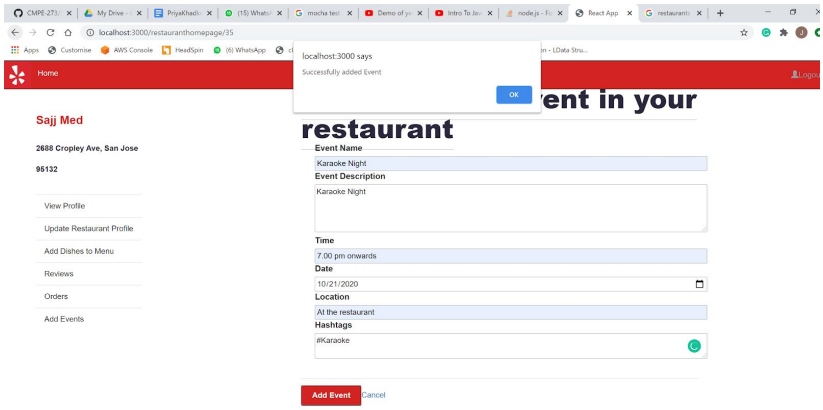


On clicking the customer's name, the owner is able to view the customer's profile

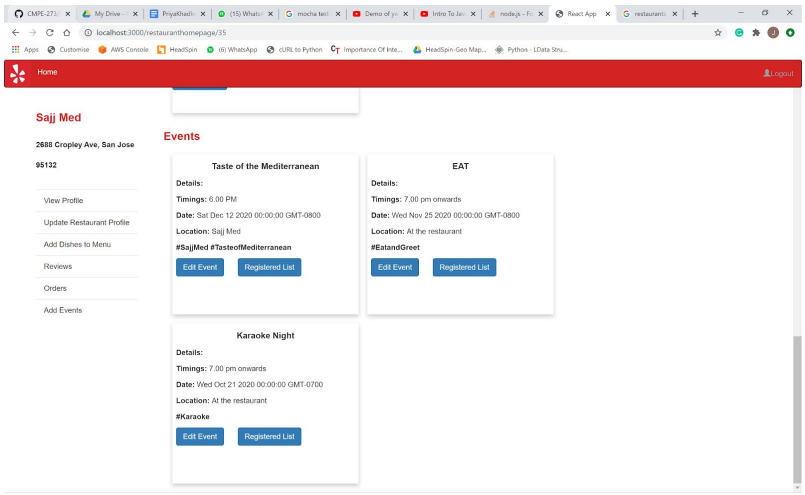


6)**Add Events**

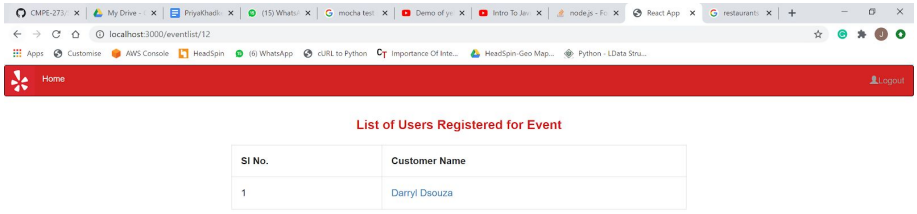
The restaurant owner should be able to add an event



The added event is reflected in the profile



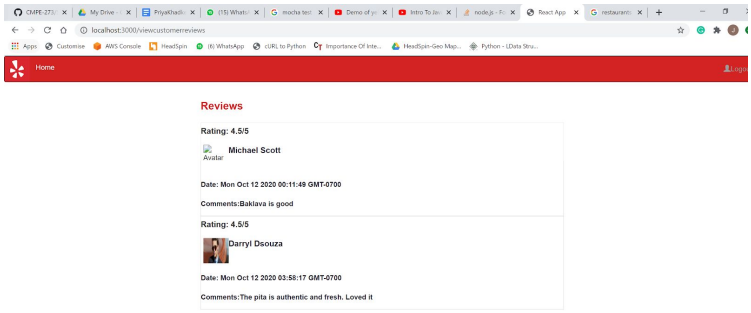
The owner should be able to see the people registered for each event

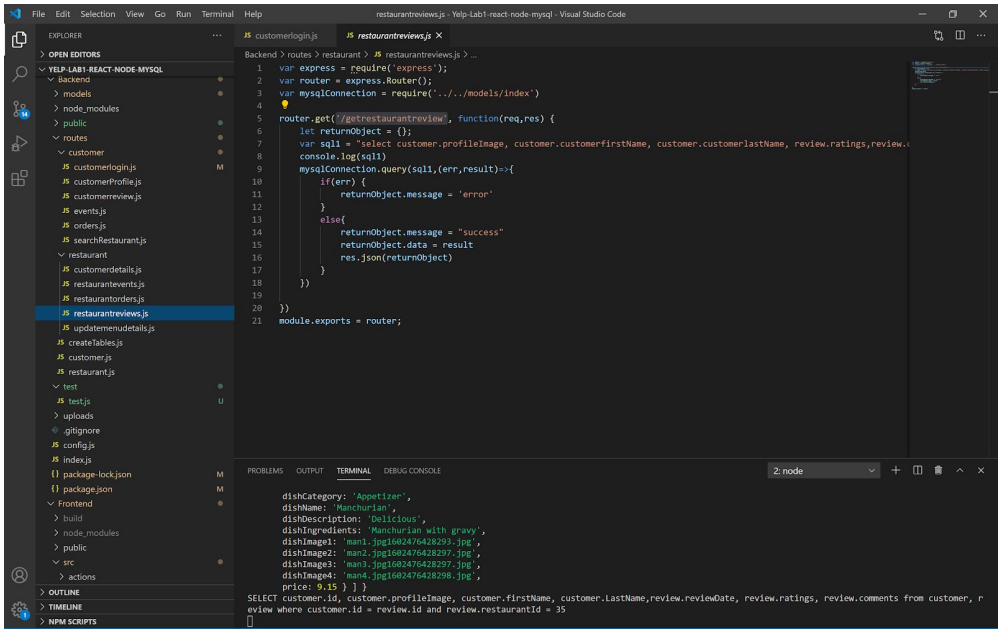


If the owner clicks on the user's name he will be redirected to the user's profile(Same as in case of orders)

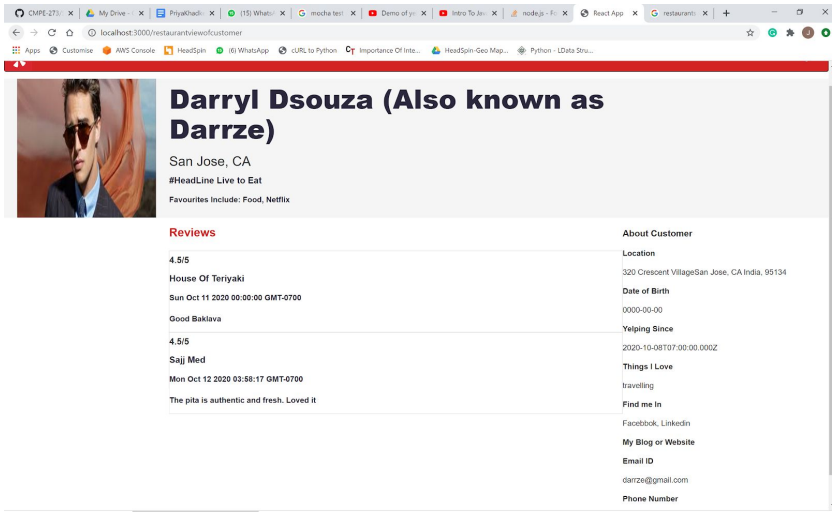
7)Reviews:

The user should be able to see the reviews of customers for his restaurant. If clicked on the user's name, the restaurant owner should be able to see the user's profile.





On clicking the restaurant owner should be able to view the profile of the user



Performance & Testing:

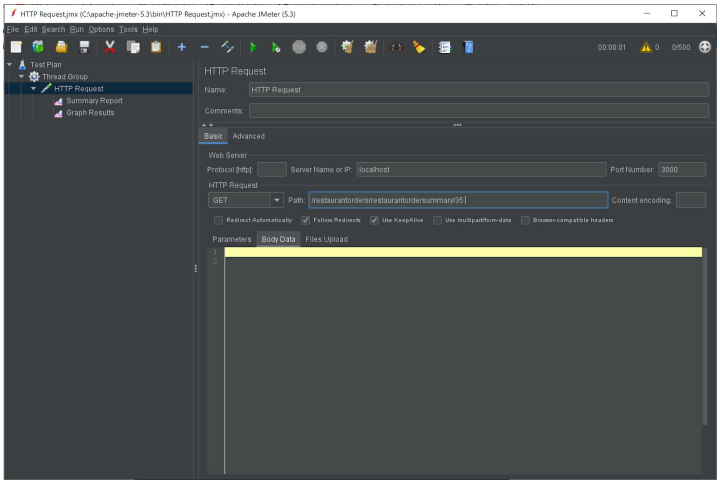
Observations from JMeter and load testing

3 APIs were tested for performance using JMeter which were /customer login, search restaurant and view orders

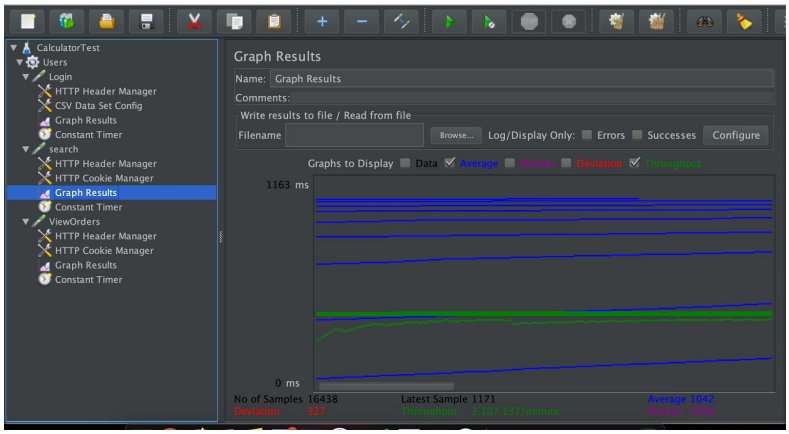
1. The yelp application's REST API's were tested against 100 to 500 concurrent users. The API's successfully returned the data even with 500 concurrent users and no errors were seen either with or without connection pooling
2. It was observed that the performance graph was better with pooled connections as the average response time was lower when compared to a non pooled connection
3. The throughput seen in connection pooling showed an increase indicating that the number of requests completed at average was at a better rate when compared to a connection without pooling

API	Users	**Average Response Time(ms) (Connection Pooling)**	**Average Response Time(ms) (without Pooling)**
GET /restaurantorders/restaurantordersummary/35	100	1042	1047
	200	1113	1453
	300	1491	2485
	400	3133	4465
	500	5617	7100
POST customerlogin/customerlogin	100	1011	4575
	200	4009	4927
	300	4010	8122
	400	4013	10253
	500	4015	12660

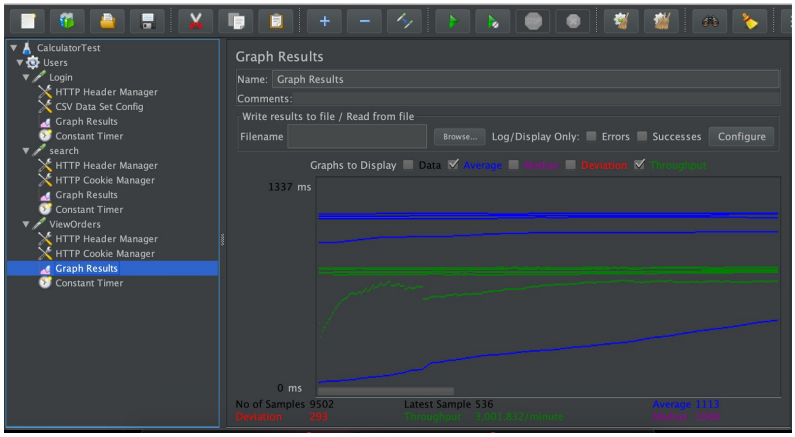
GET Request **/restaurantorders/restaurantordersummary/35 (Connection Pooling)**



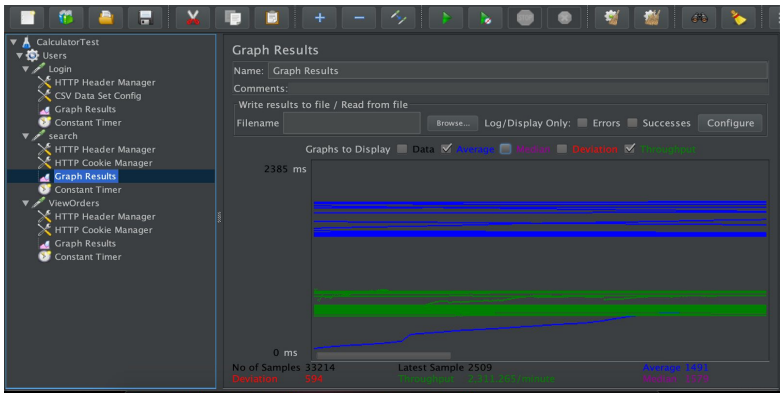
100 users



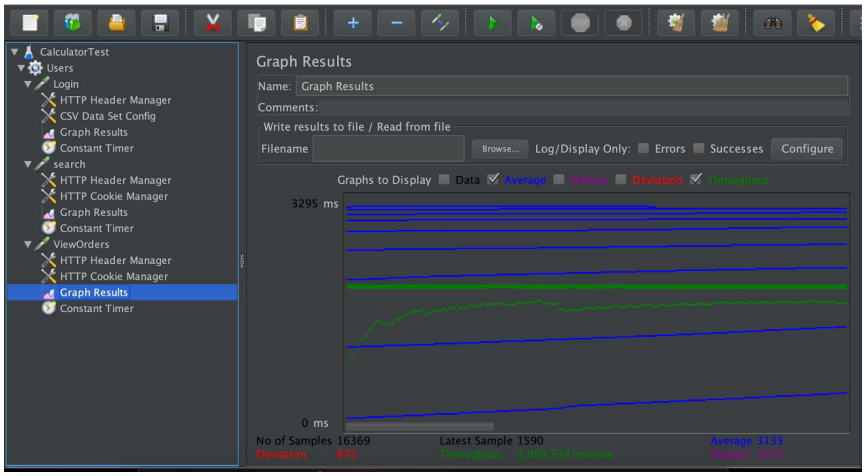
200users



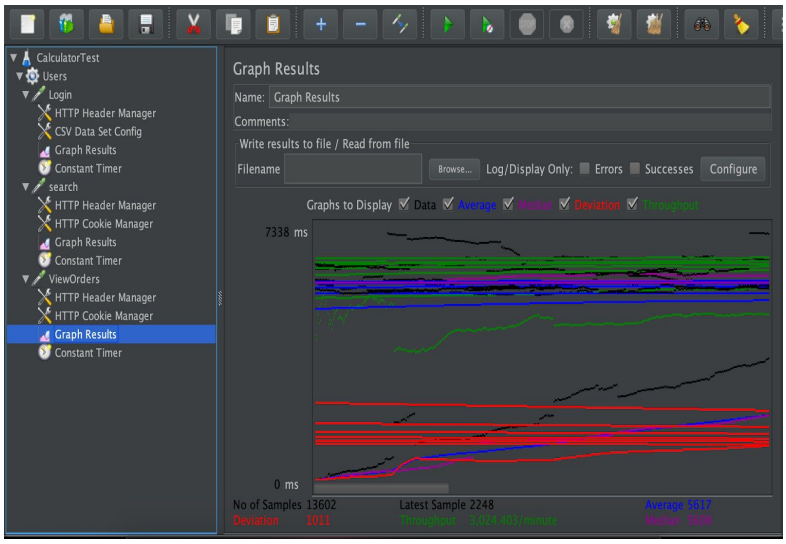
300 users



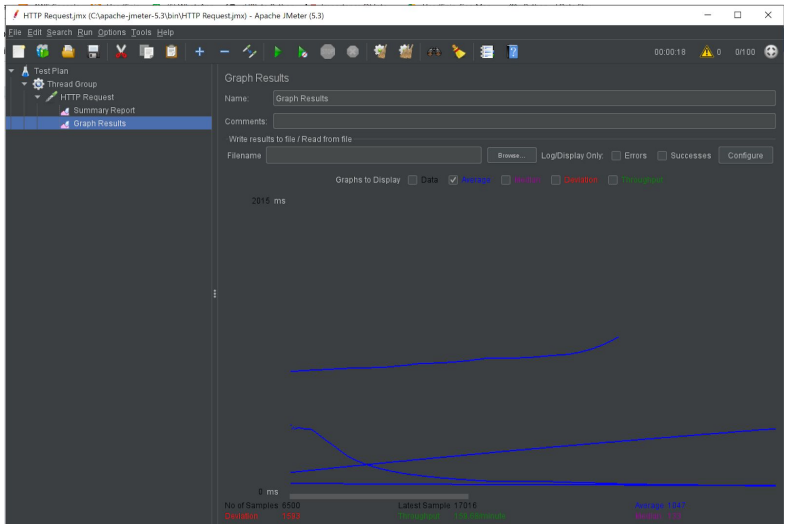
400 users



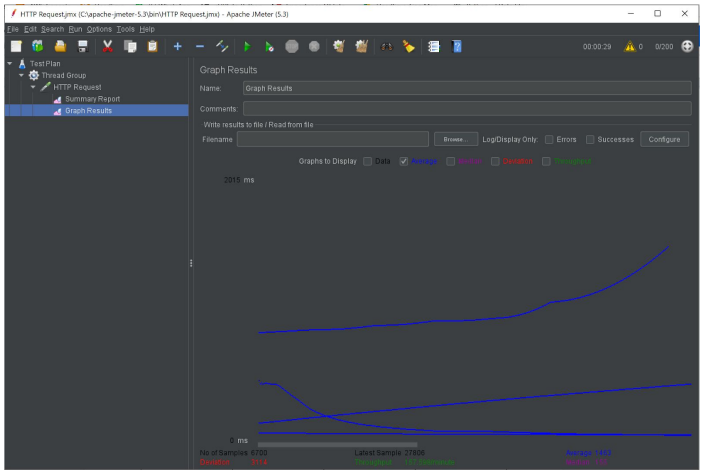
500 users



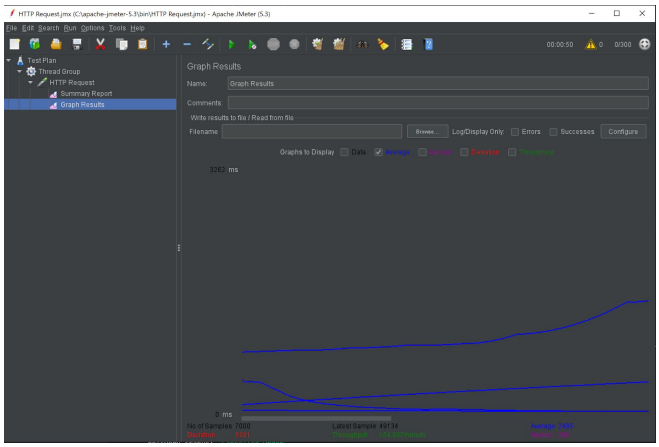
GET Request **/restaurantorders/restaurantordersummary/35 (Without Connection Pooling)**
100 users



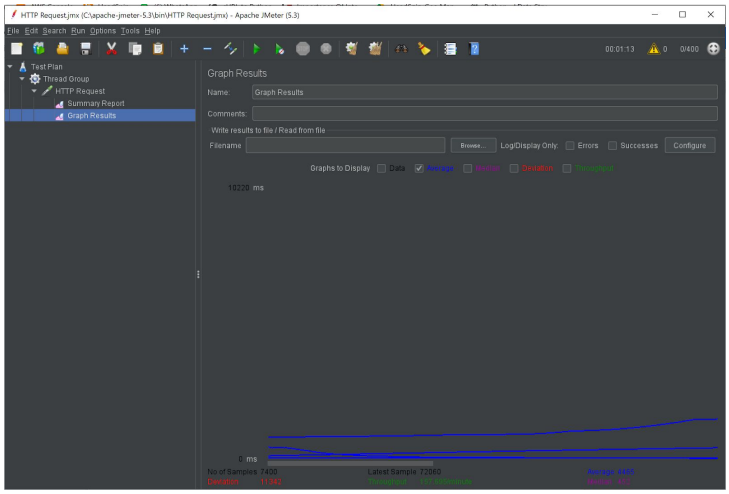
200 users



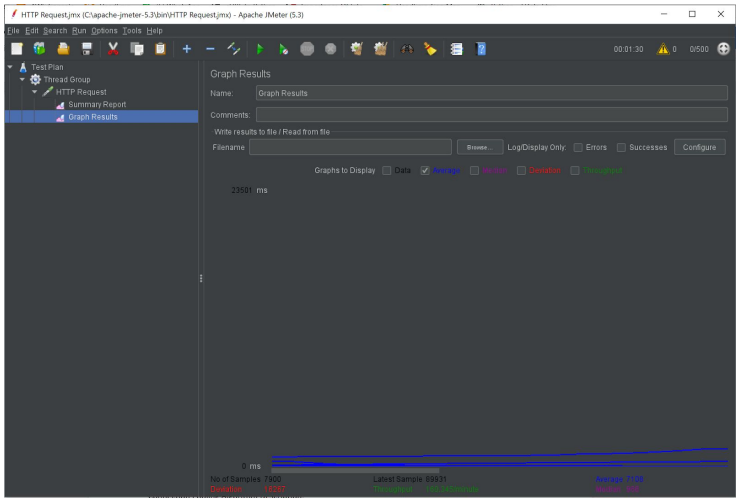
300 users



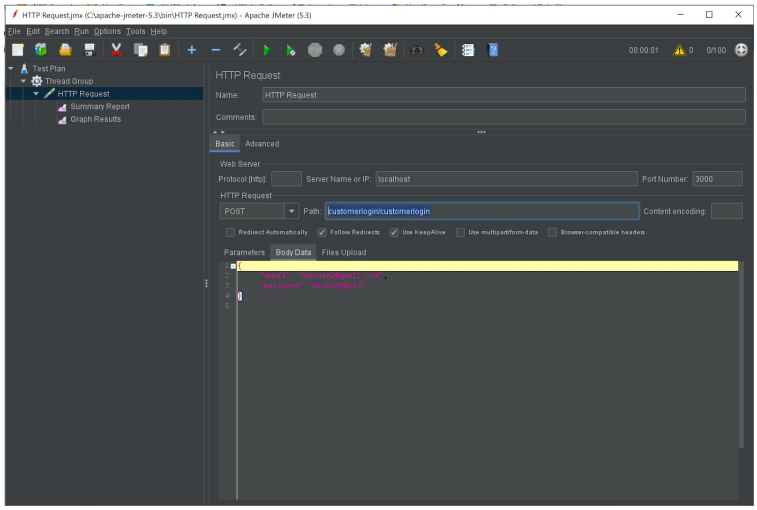
400 users



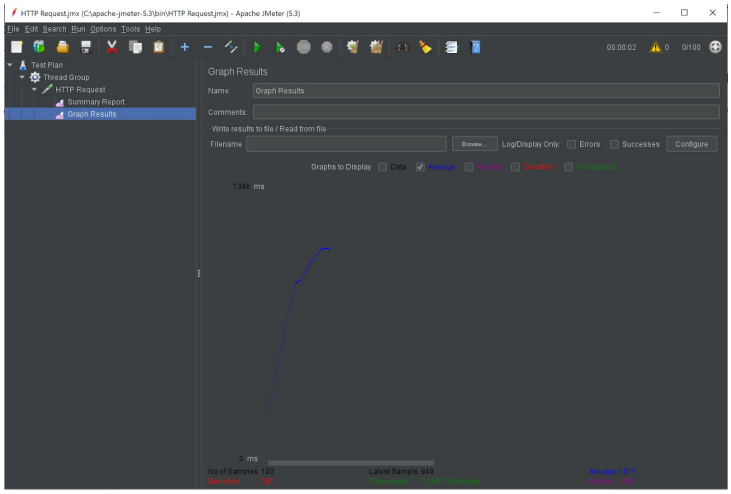
500 users



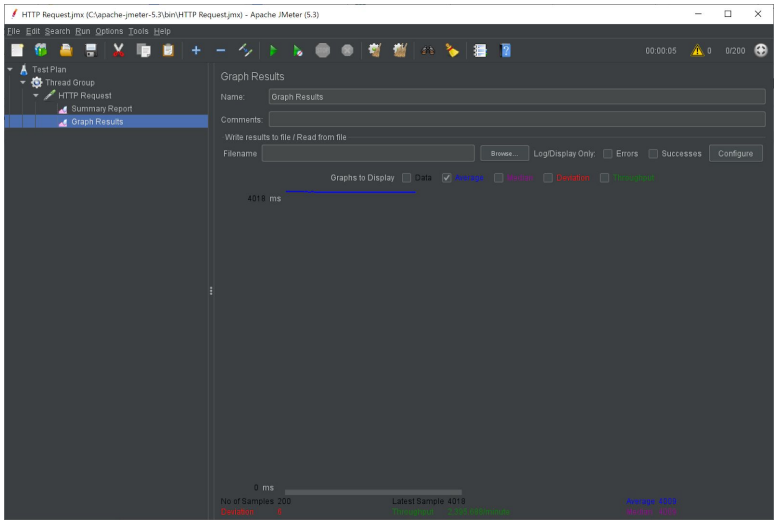
POST customerlogin/customerlogin (With Connection Pooling)



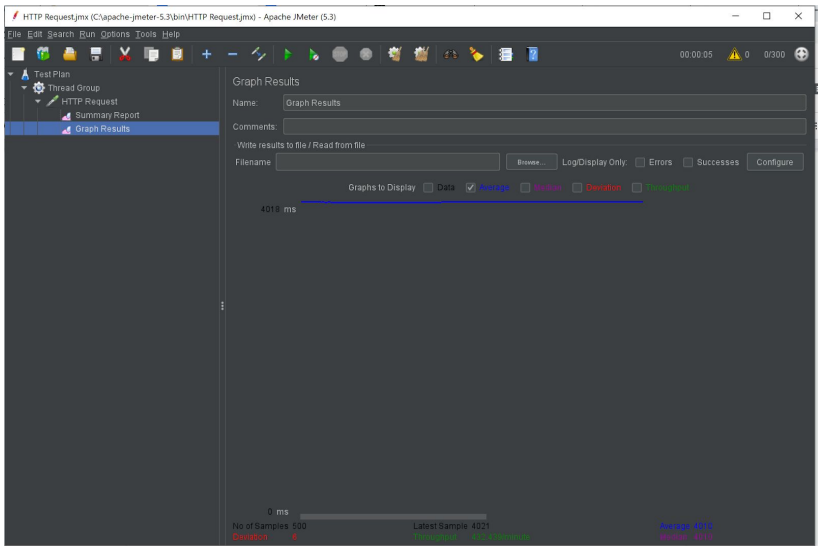
100 users



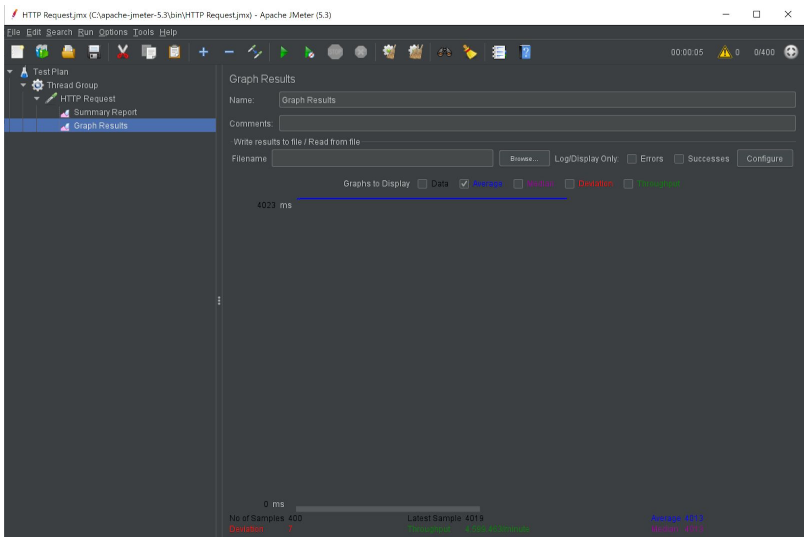
200 users



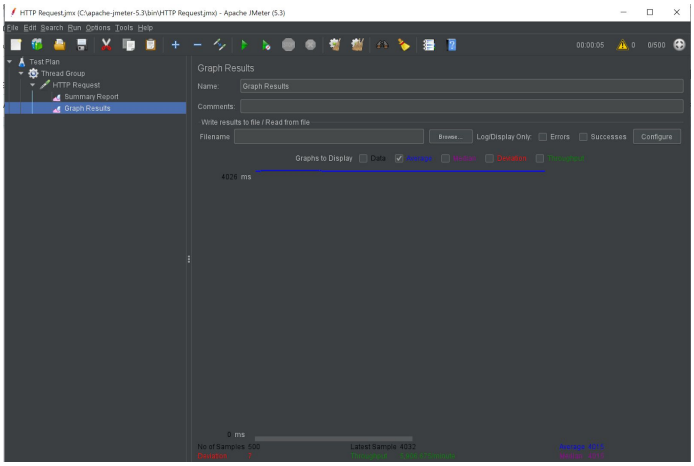
300 users



400 users

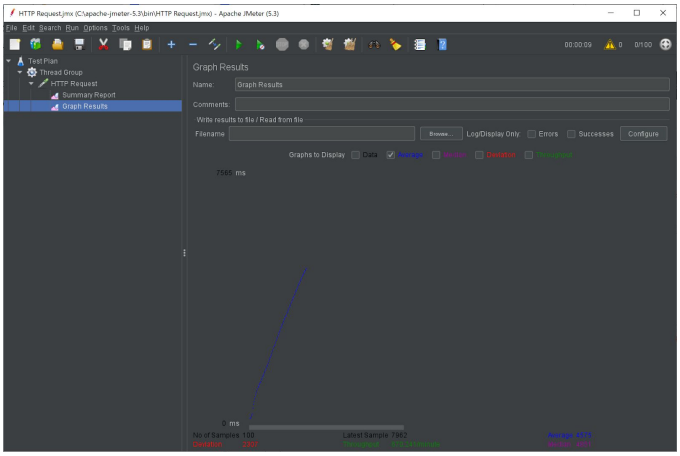


500 users

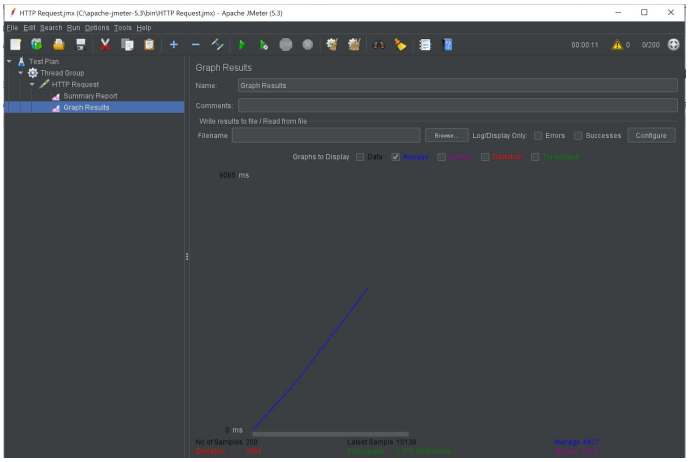


POST customerlogin/customerlogin (Without Connection Pooling)

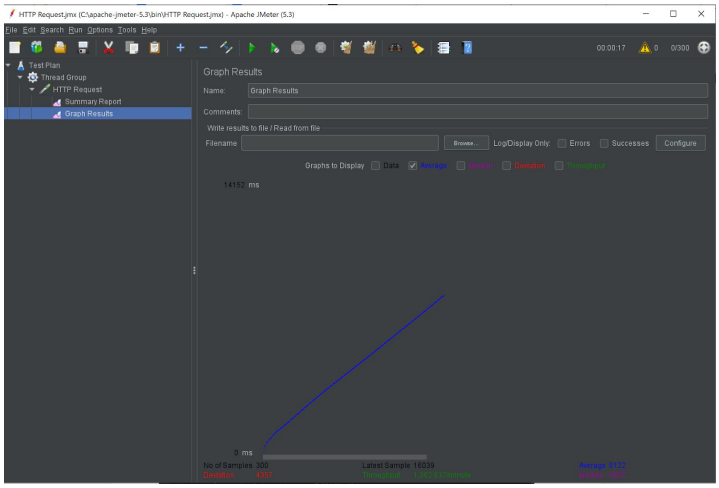
100 users



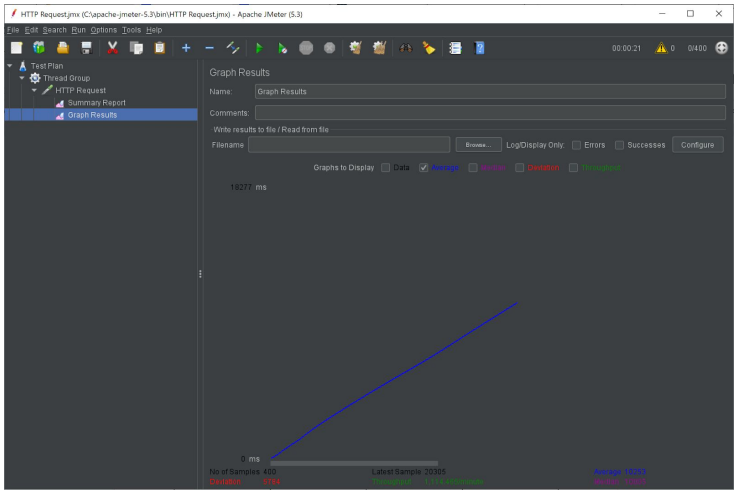
200 users



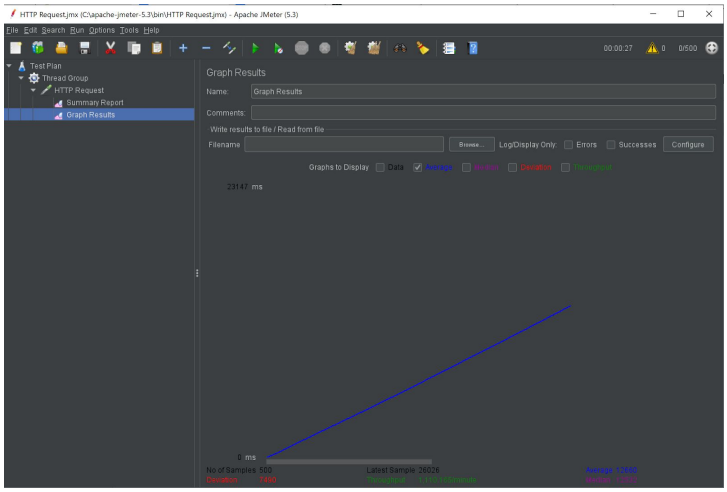
300 users



400 users



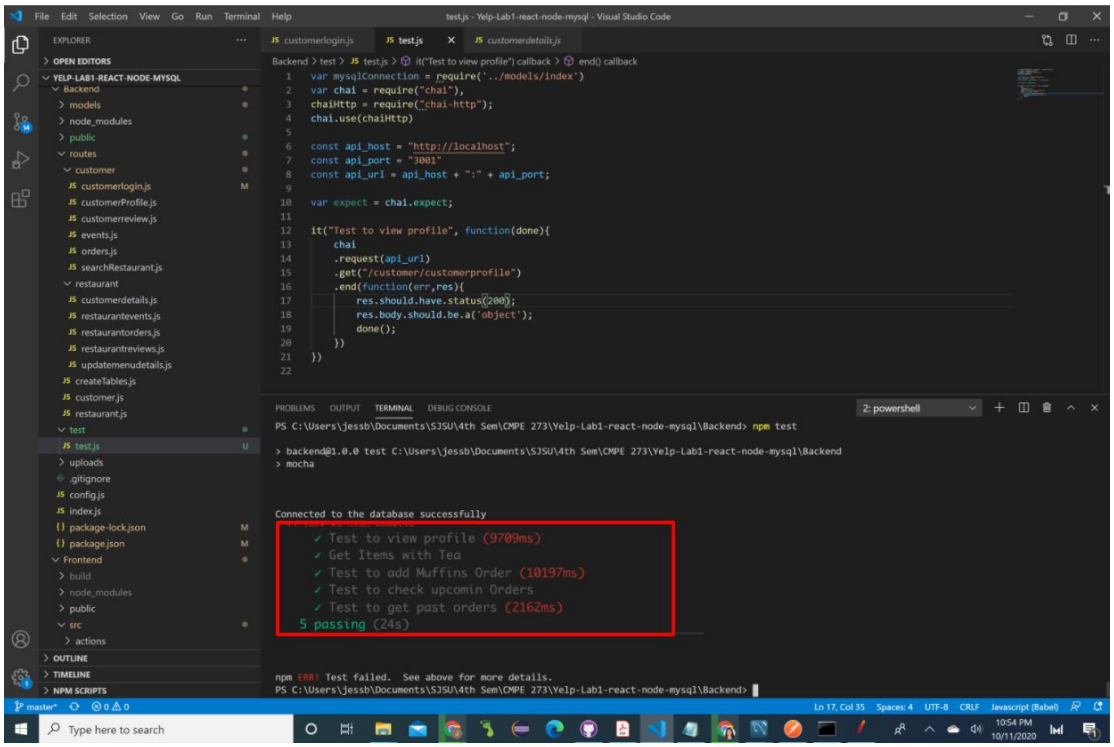
500 users



1b Mocha Testing

5 API's were randomly selected for Mocha testing

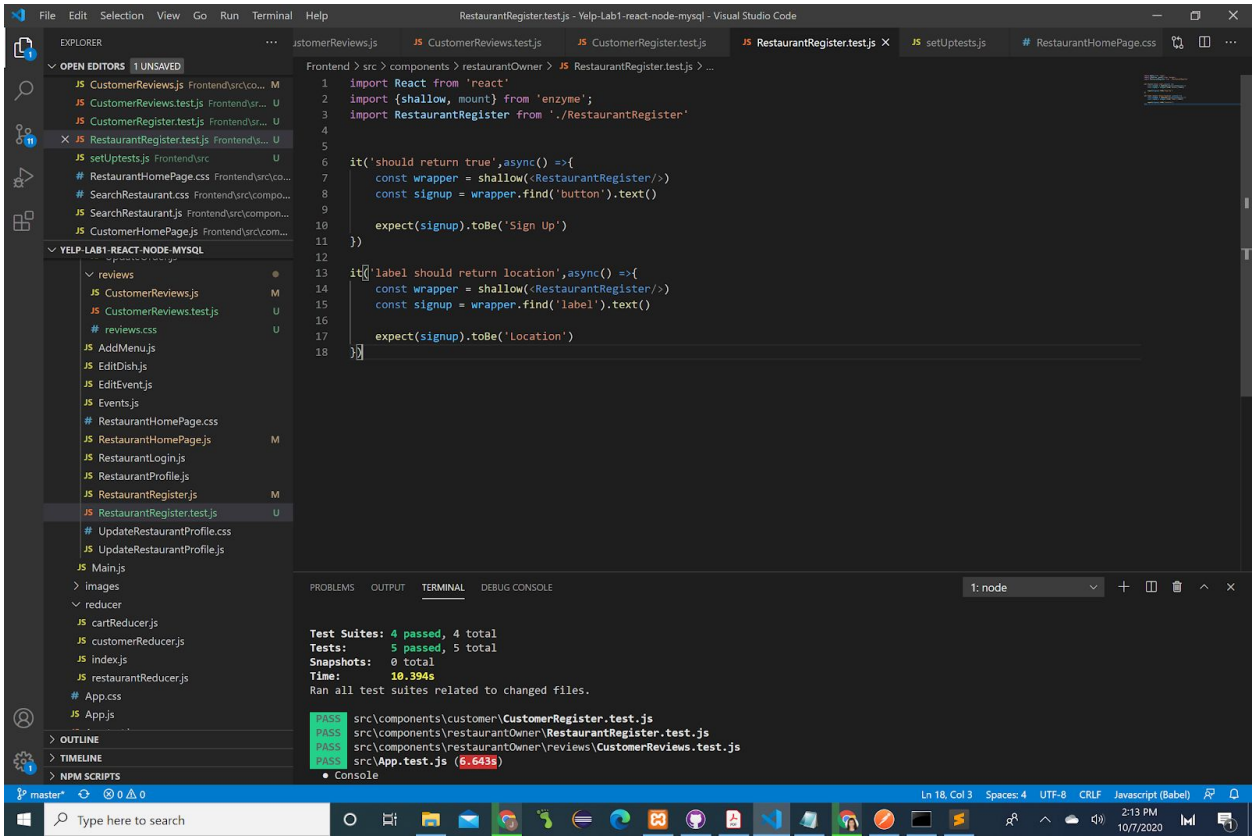
1. View customer profile
2. Get order details for dish category beverages, (Tea)
3. Customer placing an order
4. Check for newly received orders
5. Get order history for customer



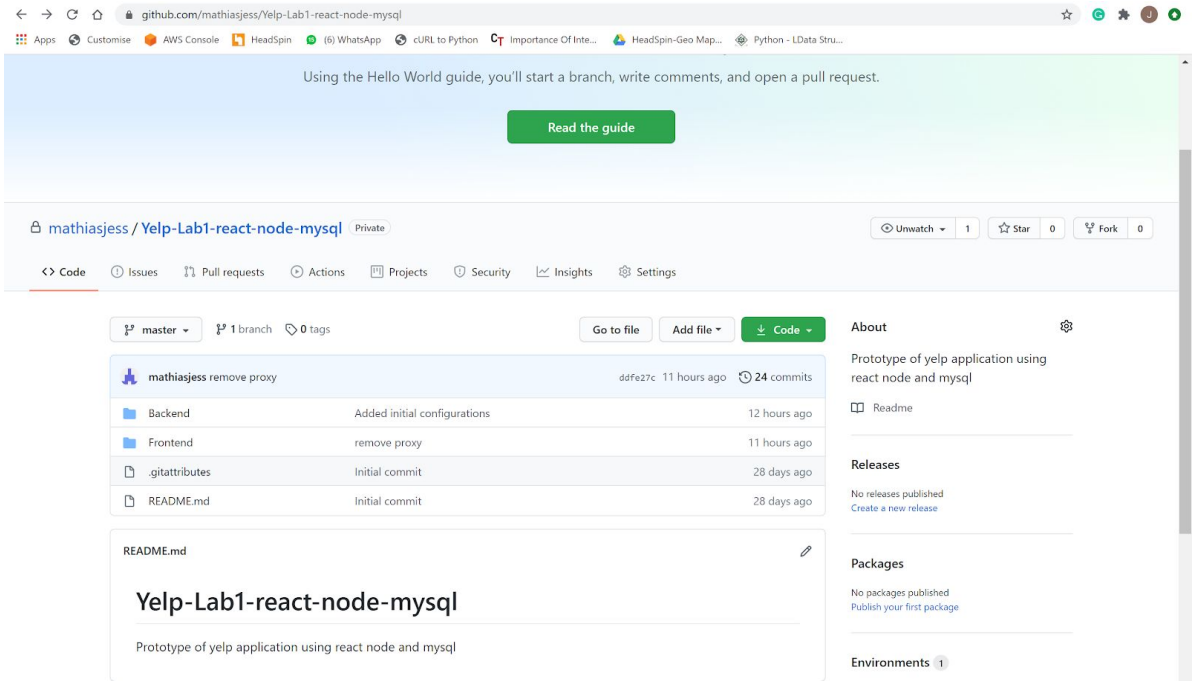
2) **Enzyme Testing**

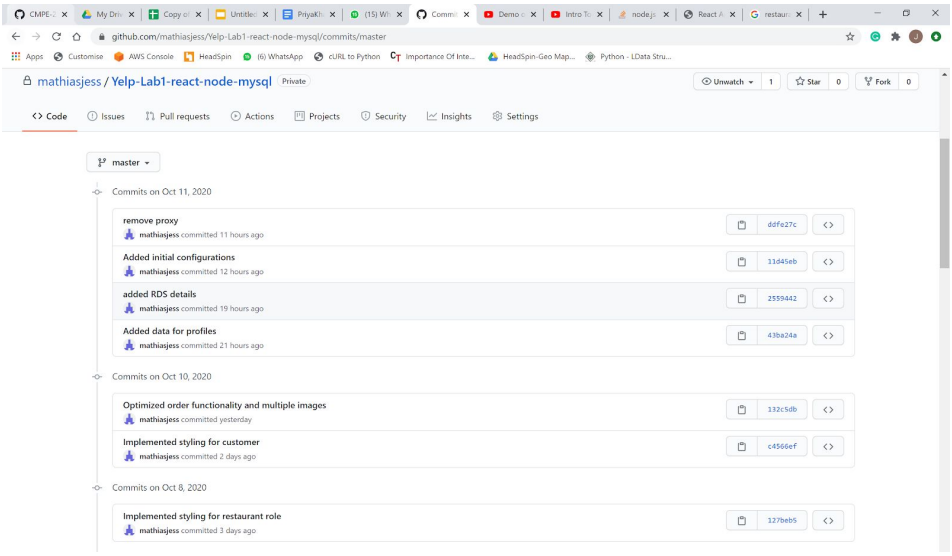
is used to test the react components in the front end. Enzyme tests were performed on 3 components namely

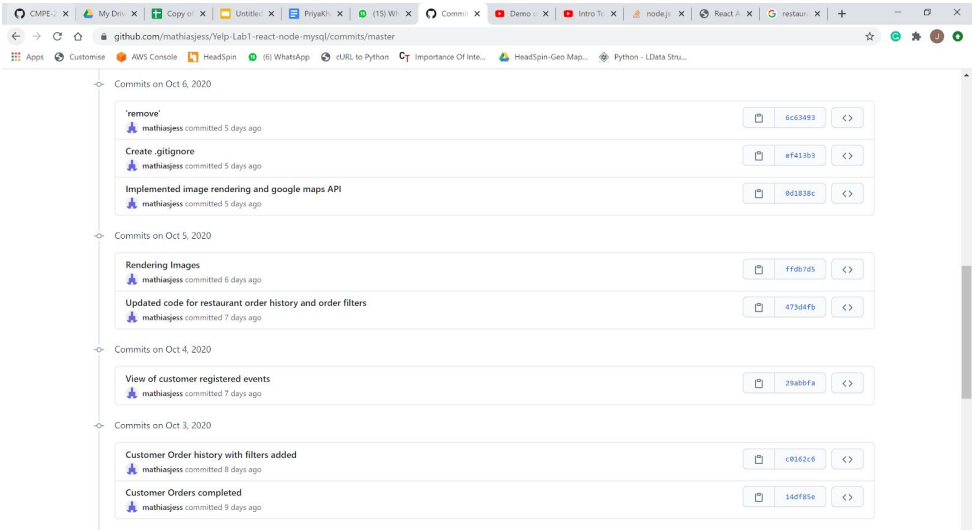
1) Customer Register
2) Customer Review
3) Restaurant Register

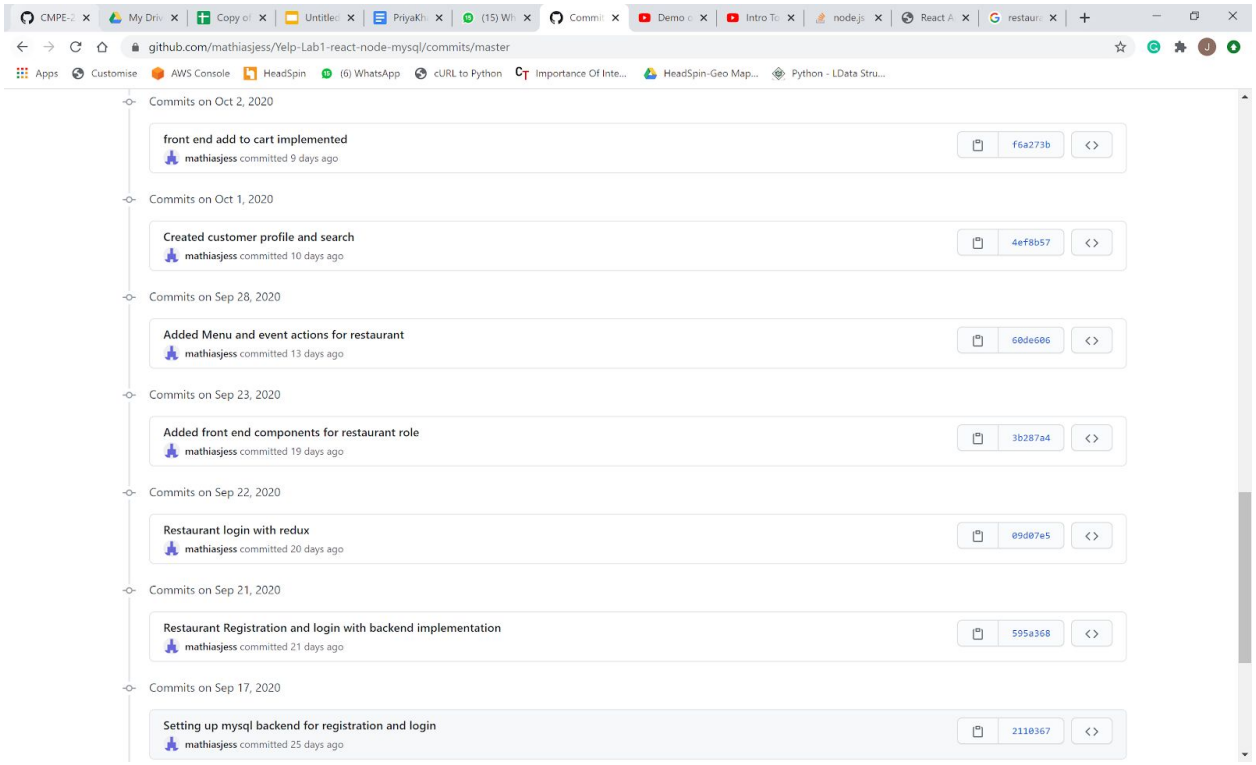


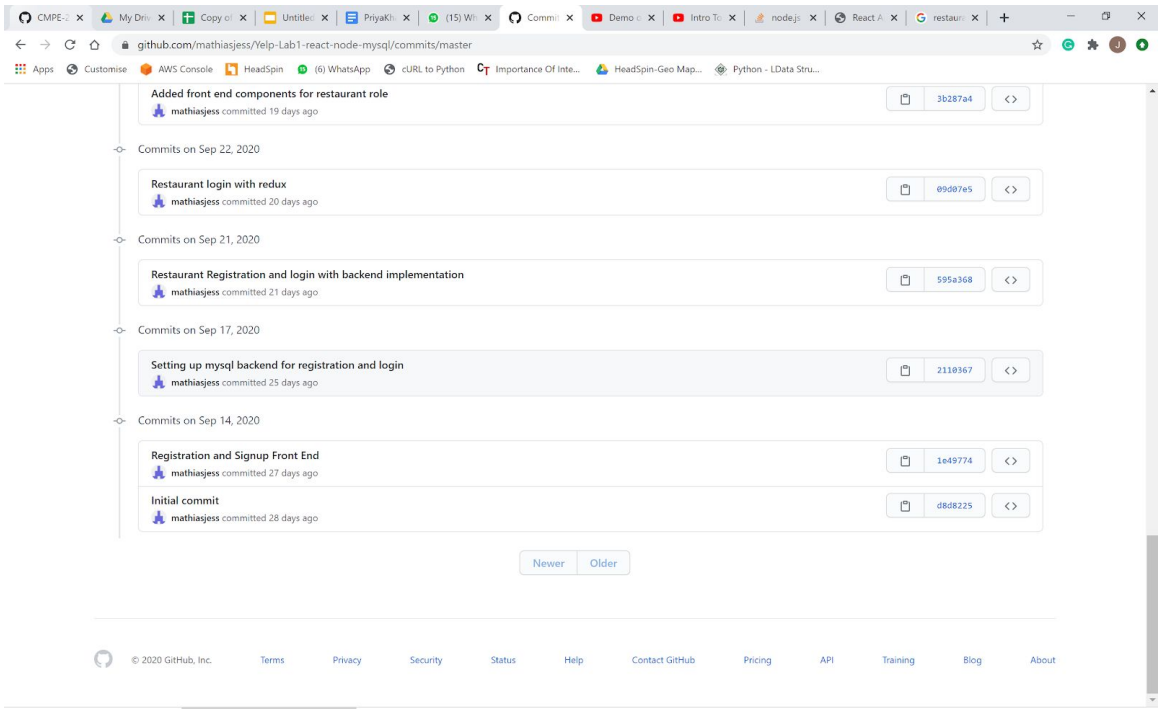
Git Commit History




github.com/mathiasjess/Yelp-Lab1-react-node-mysql/commits/master
mathiasjess / Yelp-Lab1-react-node-mysql
Private
Unwatch 1
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
master
Commits on Oct 11, 2020
remove proxy
mathiasjess committed 11 hours ago
ddfe27c
Added initial configurations
mathiasjess committed 12 hours ago
31d45eb
added RDS details
mathiasjess committed 19 hours ago
2559442
Added data for profiles
mathiasjess committed 21 hours ago
43ba24a
Commits on Oct 10, 2020
Optimized order functionality and multiple images
mathiasjess committed yesterday
132c5db
Implemented styling for customer
mathiasjess committed 2 days ago
c4566ef
Commits on Oct 8, 2020
Implemented styling for restaurant role
mathiasjess committed 3 days ago
127beb5


github.com/mathiasjess/Yelp-Lab1-react-node-mysql/commits/master
Commits on Oct 6, 2020
'remove'
mathiasjess committed 5 days ago
6c63493
Create .gitignore
mathiasjess committed 5 days ago
ef413b3
Implemented image rendering and google maps API
mathiasjess committed 5 days ago
0d1838c
Commits on Oct 5, 2020
Rendering Images
mathiasjess committed 6 days ago
ffdb7d5
Updated code for restaurant order history and order filters
mathiasjess committed 7 days ago
473d4fb
Commits on Oct 4, 2020
View of customer registered events
mathiasjess committed 7 days ago
29abbfa
Commits on Oct 3, 2020
Customer Order history with filters added
mathiasjess committed 8 days ago
c0162c6
Customer Orders completed
mathiasjess committed 9 days ago
14df85e





Questions

1) Compare the results of graphs with and without in-built mysql connection pooling of database. Explain the result in detail and describe the connection pooling algorithm if you need to implement connection pooling on your own.

API	Users	Average Response Time(ms) (Connection Pooling)	Average Response Time(ms) (without Pooling)
GET /restaurantorders/restaurantordersummary/35	100	1042	1047
	200	1113	1453
	300	1491	2485
	400	3133	4465
	500	5617	7100
POST customerlogin/customerlogin	100	1011	4575
	200	4009	4927
	300	4010	8122
	400	4013	10253
	500	4015	12660

As seen from above table, the average response time for each of the three API was improved by max 58%.With connection pool size as 50, response time was improved. Even with increasing load conditions this performance improvement was seen throughout the load test. From this, it can be concluded that making a new DB connection with every request is indeed a costly operation, with increasing hits/sec this may affect the response time of the application.

Connection Pooling Algorithm Description:

Min Pool size , Max Pool size , datasource

```
CONNECT_POOL:
        GET datasource
        GET min_pool_size
        GET  max_pool_size
        DO:
            POLL DB connection request

        IF ( request is get DB connection)
                IF ( active_connections < max_pool_size)
                  RE-USE the existing open connection from pool ;
        IF ( request is close DB connection)
                Store the connection in the Pool without closing it.
```

2. What are the ways to improve SQL Performance? List at least 3 strategies. Explain why those strategies improve the performance.

SQL performance mainly depends on the way the queries are written and how they optimize data fetching. The three strategies to improve SQL performance are

1) **Select the required fields instead of selecting all the data with a ***

Generally when the users need to do exploratory analysis, they end up selecting * from databases. If the number of rows are limited this might not affect performance much. The issue arises when the table has many rows and columns. When the user selects *, there will be many unnecessary columns such as phone number, age, gender etc returned along with the necessary data. This will slow down the performance of the query

2) **Avoid the use of selecting DISTINCT while querying**

The select DISTINCT functionality, groups data by columns. The problem here is, DISTINCT requires a lot of processing power and slows the query down. A better way to select unique values is to select more fields that match a criteria

3) **Fetching data from two or more tables should be done with an inner join**

When fetching data from two or more tables, a where clause between each table decreases the performance of the query. Where creates a cross join which can become problematic while fetching billions of records. An inner join is more efficient in fetching the required fields.