Jessica Mathias Lab Report 2 - Yelp-Prototype-Redux-MongoDB-Kafka

Youtube Link : https://youtu.be/x3qwBG6ld2I

Github Link: https://github.com/mathiasjess/Yelp-Lab2-Redux-Node-Kafka-MongoDB

1. Introduction

The objective of lab 2 is to build on the yelp prototype application created in lab 1. This lab is designed to use redux to showcase the change of state in every component. MongoDB which is a non-relational DB is used as the database to store customer and restaurant data. Kafka messaging system is used as the middleware to stream messages between the frontend and backend. The application is hosted on AWS EC2. This lab focuses on using pagination, and sorting of results for the current functionalities, adding “all users and following” for a customer and messaging system for a restaurant and customer

2. System Design

The system design is described as a combination of assumptions, functionalities, architecture and technology

A detailed description of the application is given as follows: -

Assumptions

- Every user and restaurant owner has a unique ID and role assigned to them
- Images are stored in the backend and the name of the image is stored in the database
- All API’s are validated using passport and JWT
- Bcryptjs is used to encrypt passwords
- Every input field for the customer and restaurant has proper validations
- MongoDB with pooling as a connection is used
- ESLint is used to beautify the code

Functionalities

Restaurant:

- The restaurant owner can register using his restaurant and location
- The owner can update his profile, add/edit dishes and add/edit events
- The owner can view and update all orders to his restaurant
- The owner can click and view on a customer’s profile
- The owner can message a customer

Customer

- Customer can register using his name
- The customer can update his profile image and all other basic profile details
- The customer can search for events, sort events in order of the date and register for events
- The customer can search for restaurants based on restaurant name, cuisine, type of delivery or location

- The customer can order food from the restaurant
- The customer can write reviews from the restaurant
- The customer can view the list of all yelp users, can search for another user, filter the user based on location and zipcode and follow another user
- The user can reply to a restaurant only if the restaurant messages first, the user can also see a history of messages

API's

The API's from the lab1 functionalities are not added since they are the same. Only API's for additional functionalities such as messaging and "users tab" are present here

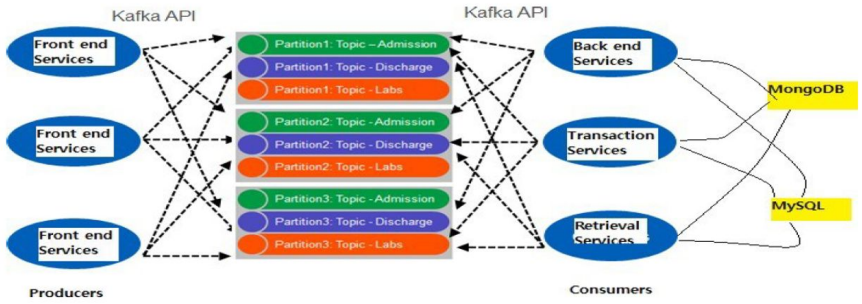
#	API	Method	Usage
1.	/getchats /getrestaurantconversations /getcustomerconversations /getallusers /getotheruserprofile	GET	To get all chats, restaurant specific Chats, customer chats, get all yelp Users and fetch profile details for Other users
2.	/addfollowing	POST	To add the customer to following list

Architecture

The architecture is split into 3 parts
FrontEnd: The front end is composed of React, Redux, HTML5, Javascript and CSS. Redux is used to store and update the state of all components

Backend: The backend consists of kafka backend and node backend. The request from the API comes to the node server, the node server then directs the request to the kafka server based on the topic. A topic was created for each of the functionality

Kafka Backend



Kafka which is a distributed streaming system, fetches the data from the producers and and adds it to the kafka cluster queue. The kafka consumer receives data, reads/writes to the DB depending on the request and the result is sent back to the node server as a callback

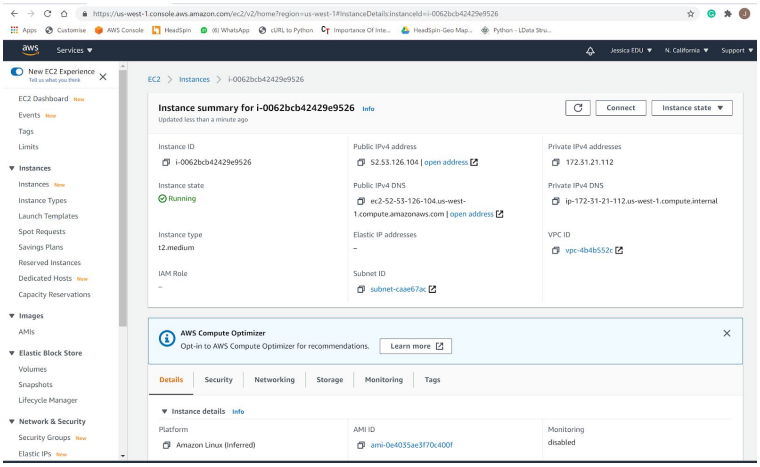
The figure above depicts Kafka 's implementation on the backend server. This is used against multiple simultaneous requests to achieve high throughput. Because of replication, it also promotes fault tolerance. The kafka cluster is handled by the zookeeper. The replication factor and partitions are set to one because of the users' limitations.

Database: The cloud version of MongoDB known as MongoDB Atlas is used to store data for the customer and restaurant. Pooling is used. Since MongoDB is a non relational database, retrieval of data is faster than relational databases

Socket.io is used to implement the messaging system between a customer and restaurant. The advantage of socket.io is that it updates the chats in real time without having to refresh the page

Deployment

The deployment of this application is done using AWS EC2.
EC2 Instance: http://ec2-52-53-126-104.us-west-1.compute.amazonaws.com:3000

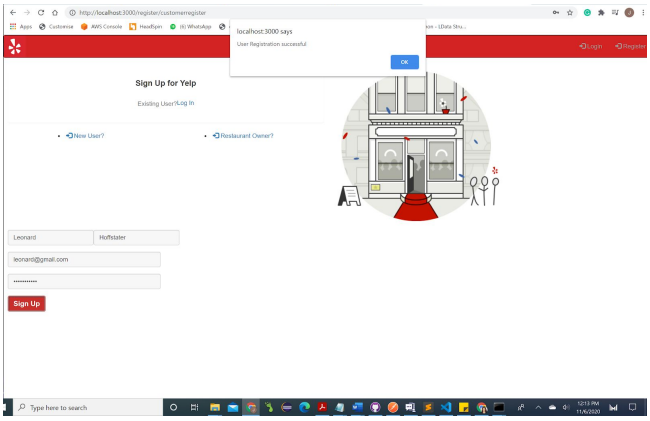


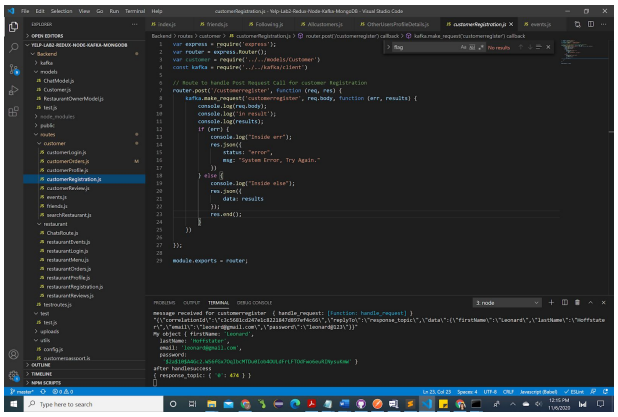
3. Results

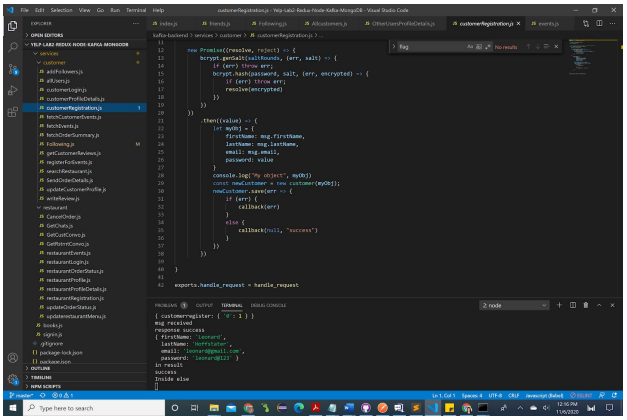
Customer Role:

1. Registration

The customer registers to the website using his first name, last name, email ID and password. Once registered, he/she is rerouted to the login page. The password is authenticated using bcrypt

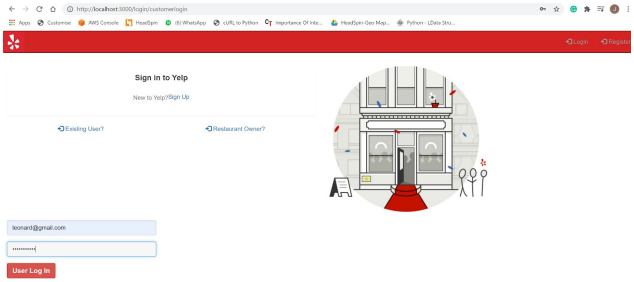


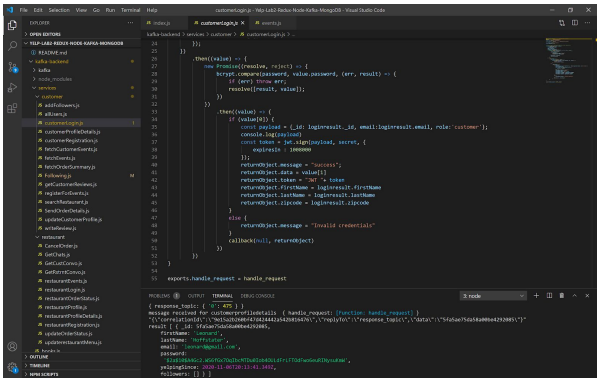


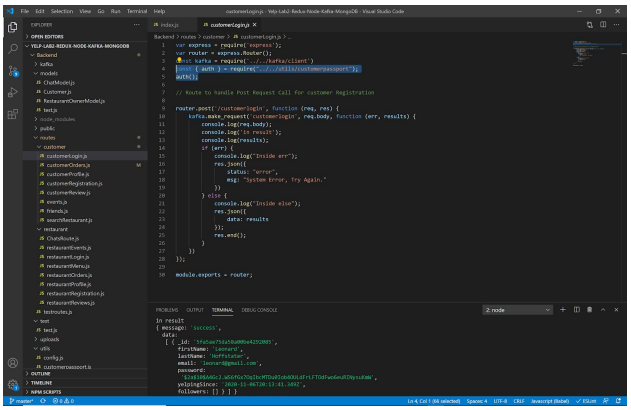


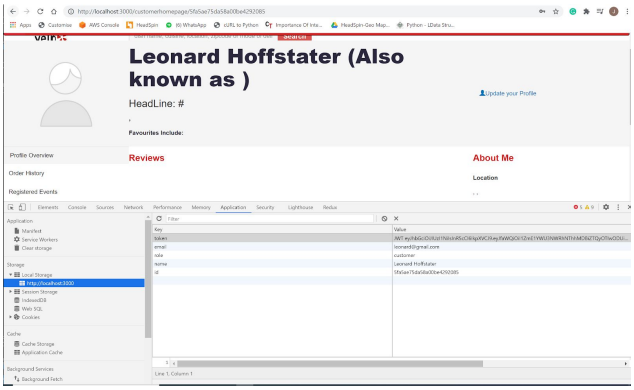
2. Login

The user logs in using his email ID and password. On successful authentication, the passport authorises the user, a JWT token is created and sent to the front end. This JWT token is used to authenticate future requests from the user.

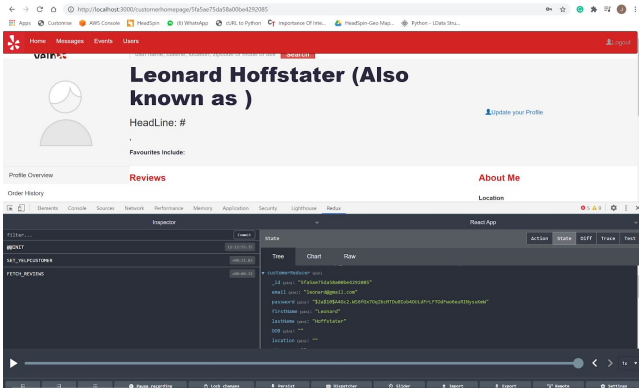






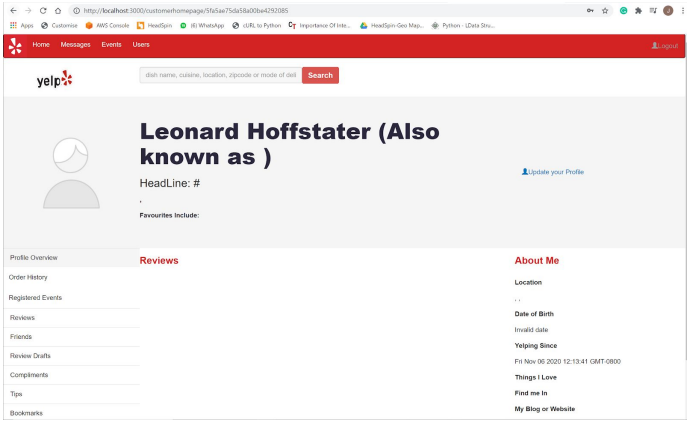


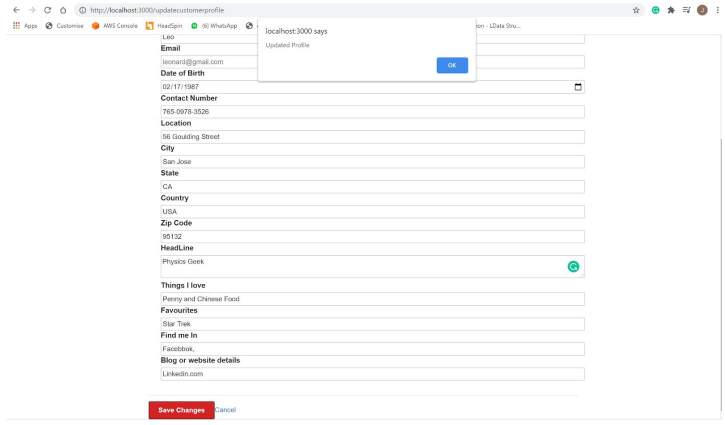
Redux is used to store the state of the user once they have logged in

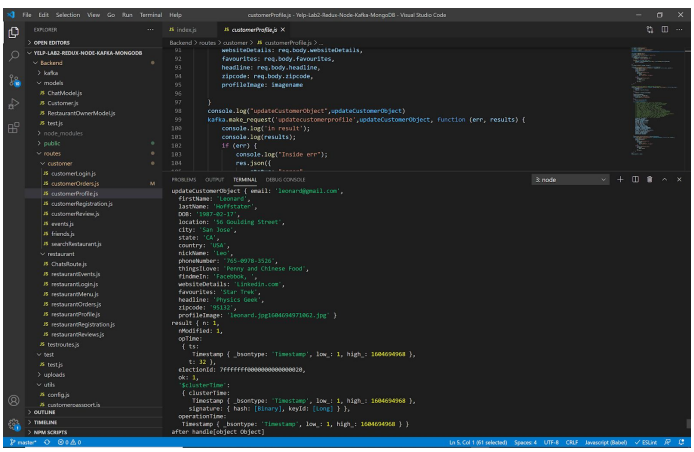


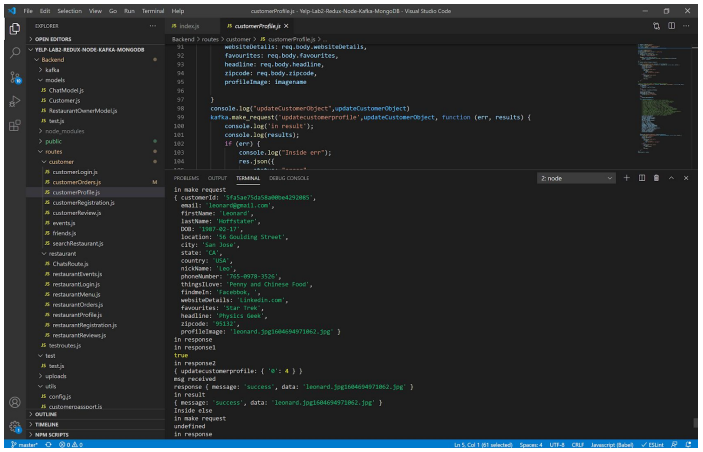
3. Profile Page

Once the user is logged in, he will be able to see his profile page. The user can then update his profile picture, basic details, his favourites, contact information and other profile details

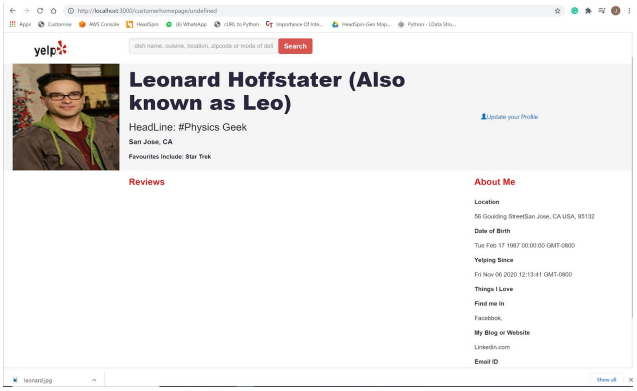


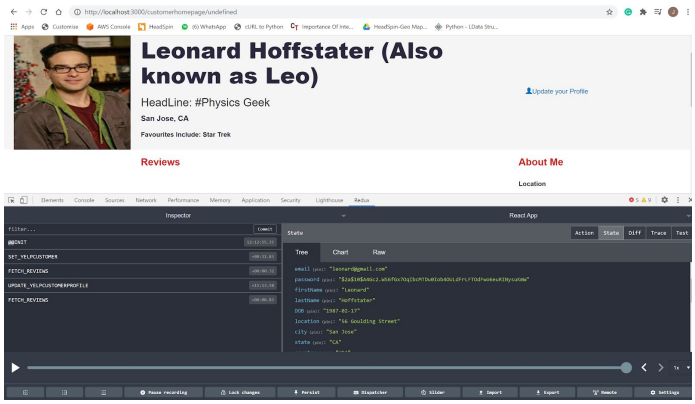






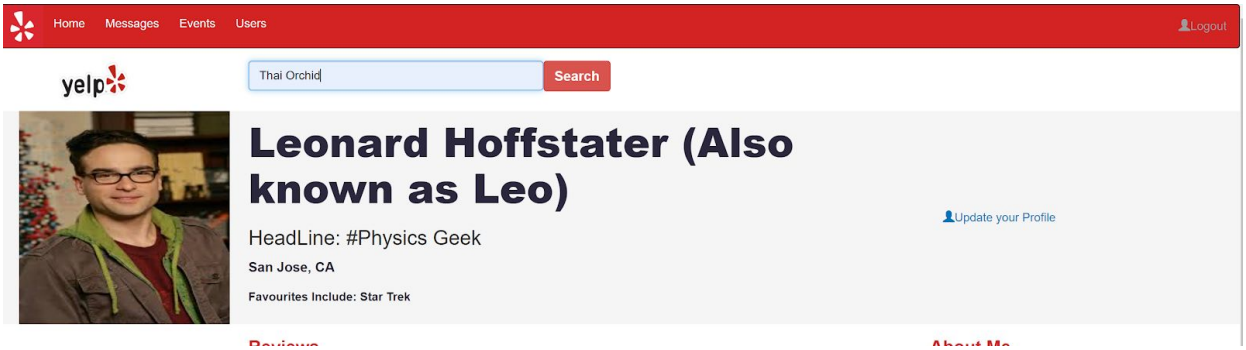
The reducer UPDATE_YELPCUSTOMERPROFILE is used to update the redux store with the user's updated profile information

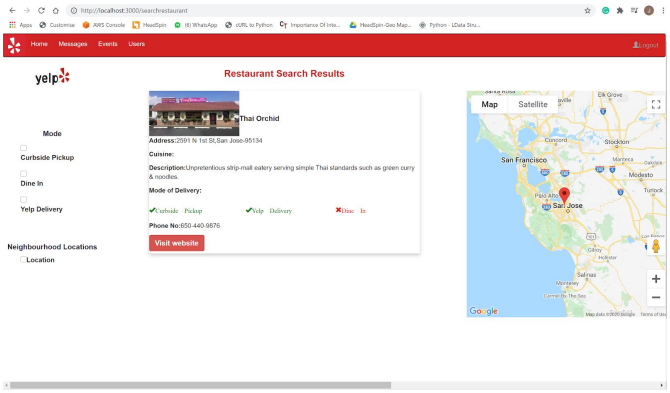




4. Restaurant Search

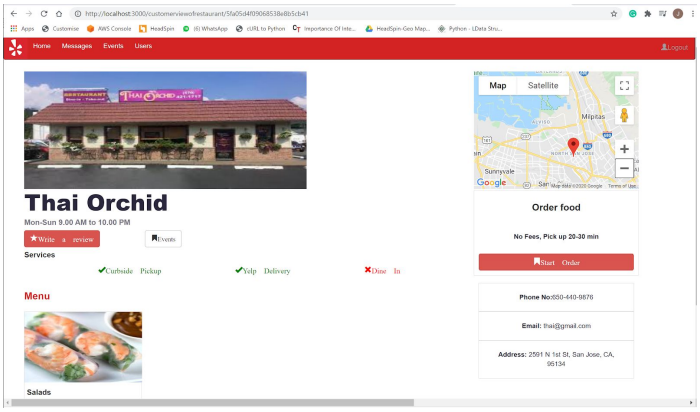
The user should be able to search for a restaurant using restaurant Name, dish Name, location, type of delivery and cuisine

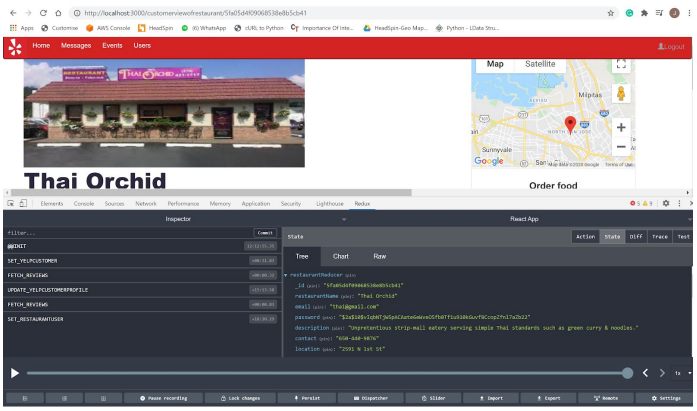




The user is further able to filter his search results based on type of delivery and neighbourhood locations. The search feature also used google maps to show the locations of the nearby restaurants

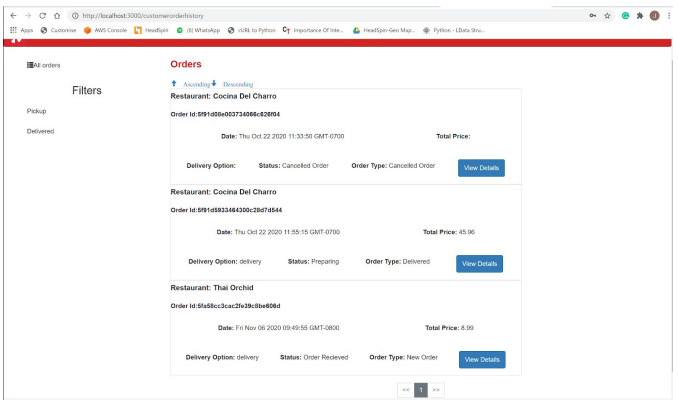
On clicking the restaurant, the user should be redirected to the restaurant's site
The redux store is updated with the restaurant's information

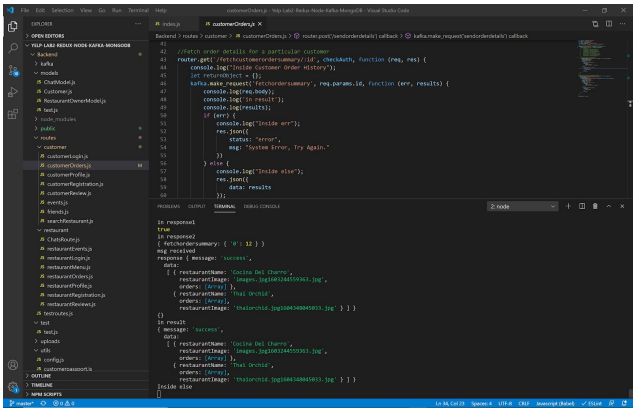




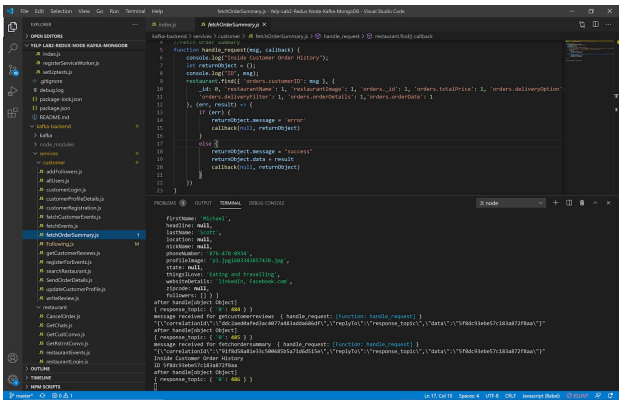
5. Orders Tab

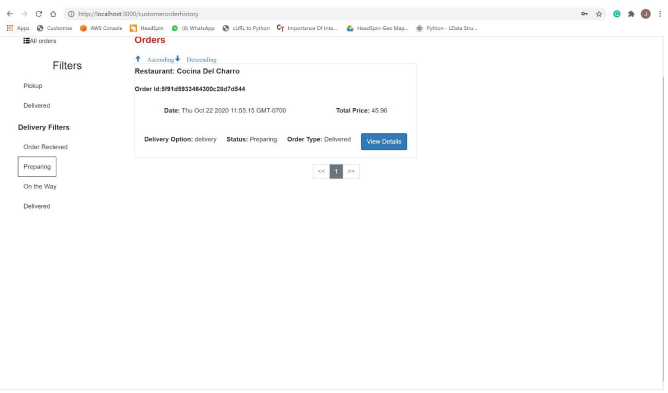
User should be able to see the order history along with the date and order details. Pagination is used to show 3 orders per page





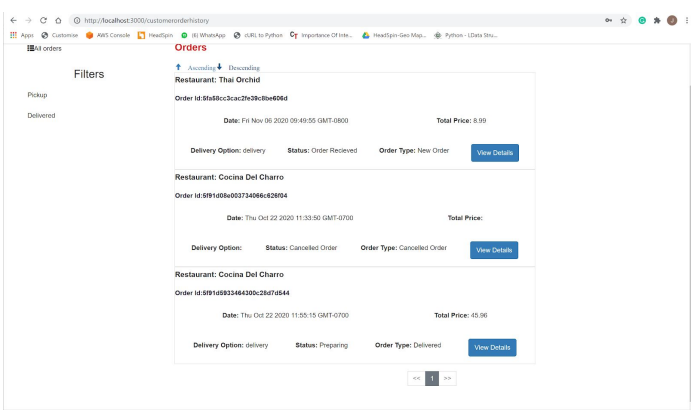
The user should be able to filter the orders based on delivery type, and delivery status(order received and order preparing)

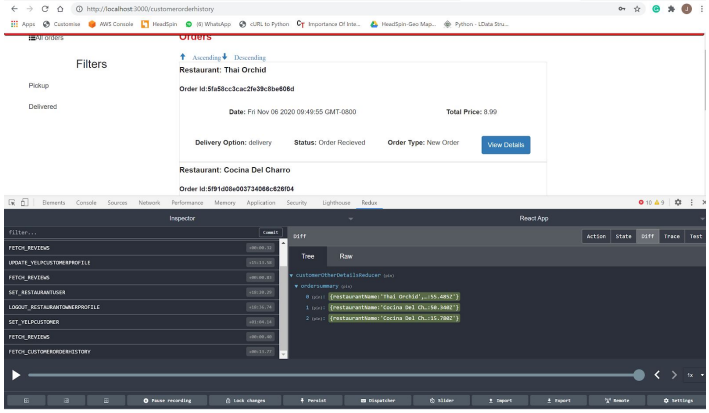




The orders can be sorted in ascending and descending order based on date

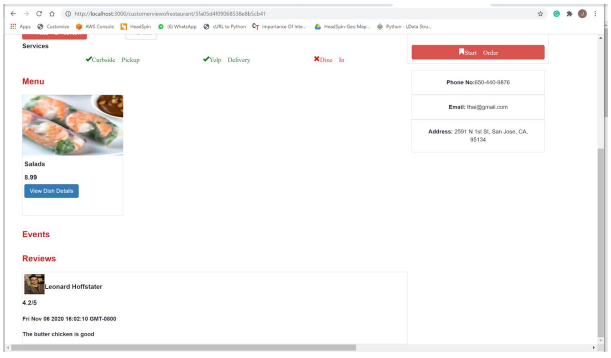
The redux store is updated to show the customer order history



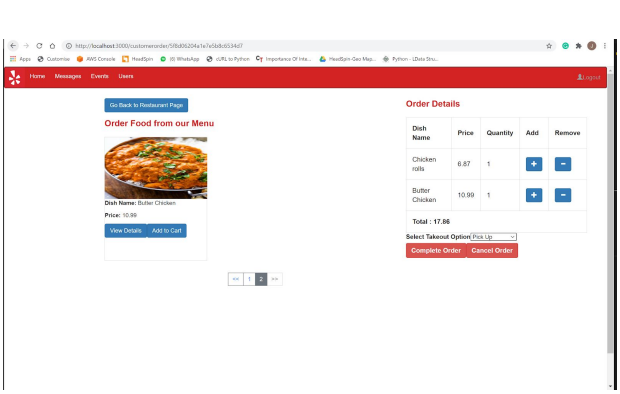
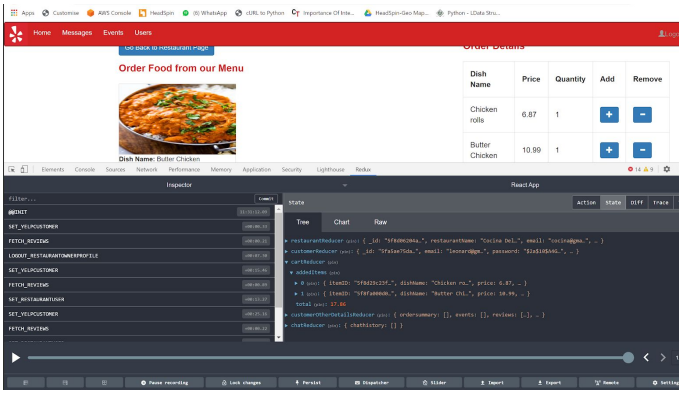
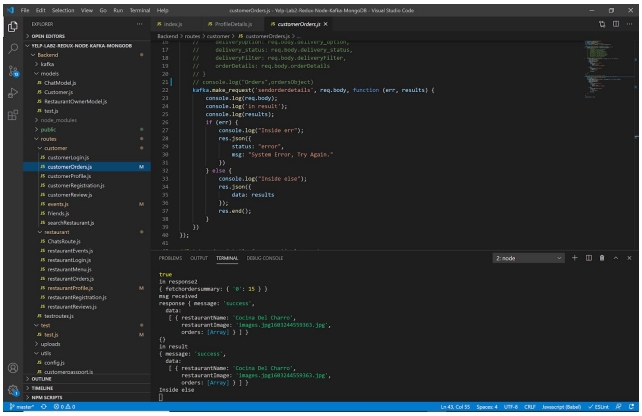
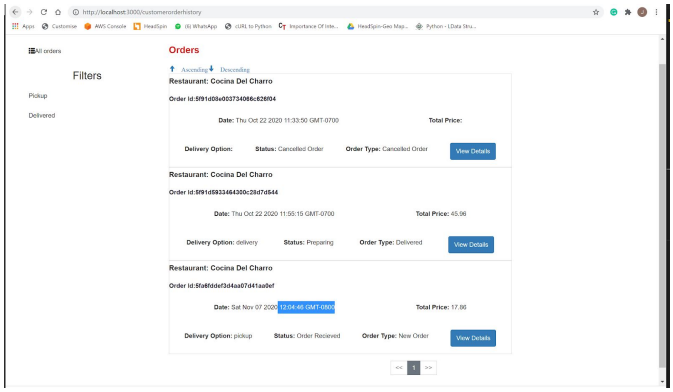


6. Restaurant Page

Once the user views the restaurant page, he can add reviews



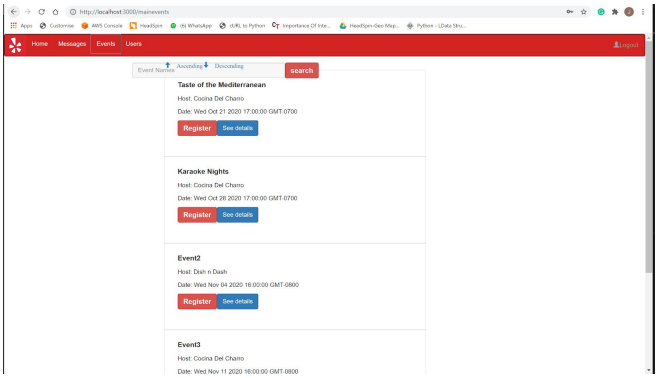
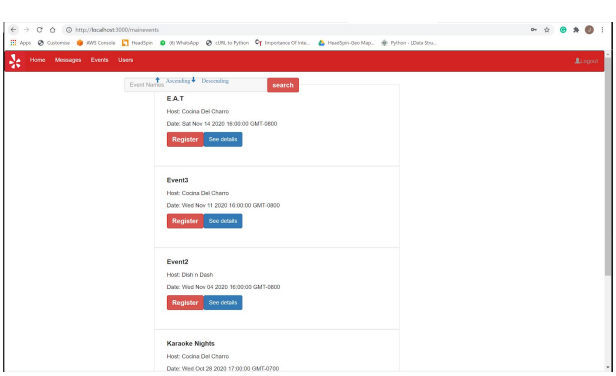
The user can choose the dishes and add to cart. Take out options can be selected as pickup or delivery. The cart handling is done by redux. Pagination is implemented.

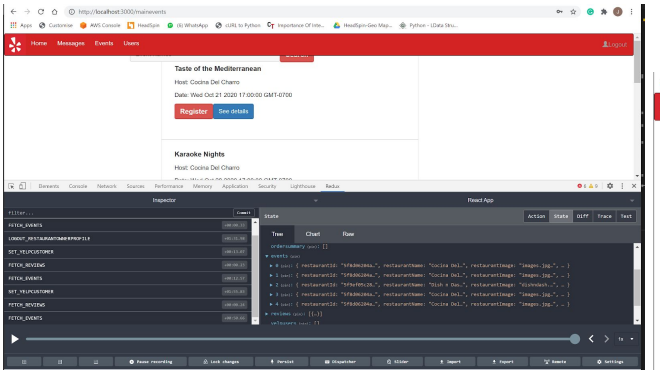
The order is successfully placed and shows up in the customer's order history

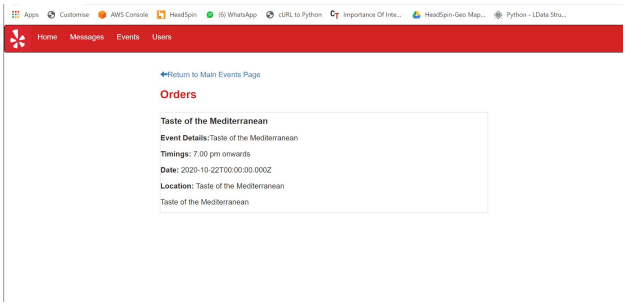
7. Events

The user can see events from all restaurants on a single page. The user can also sort the order of events in ascending and descending order of the date

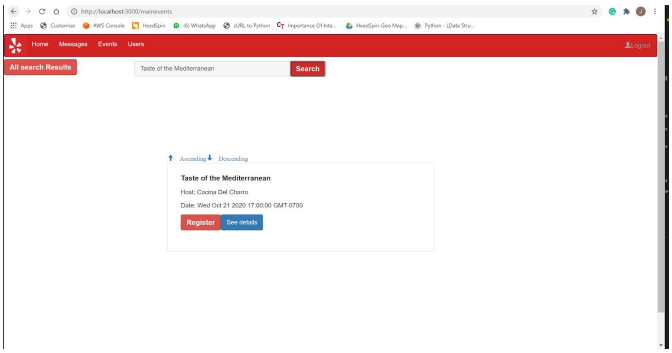



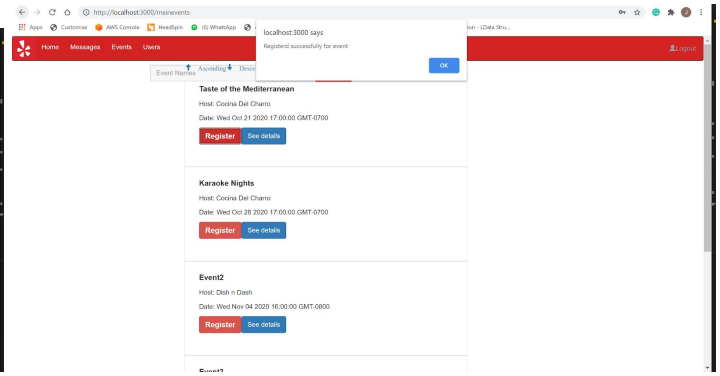
The redux store is populated with all the events
Click and view on details

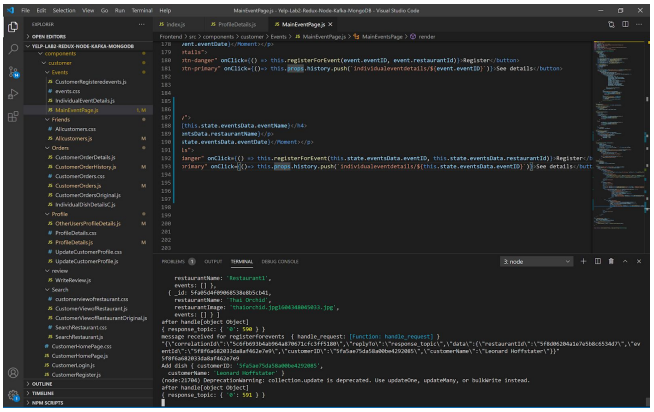


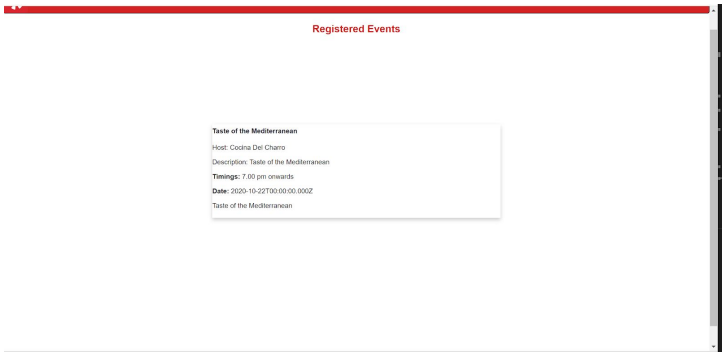


Search for a restaurant using restaurant name
Register for an event





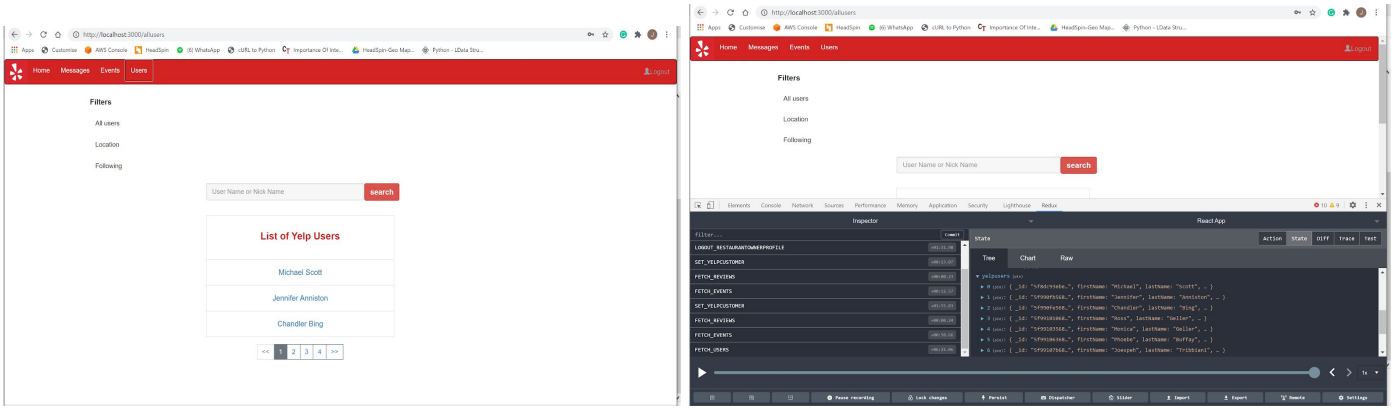




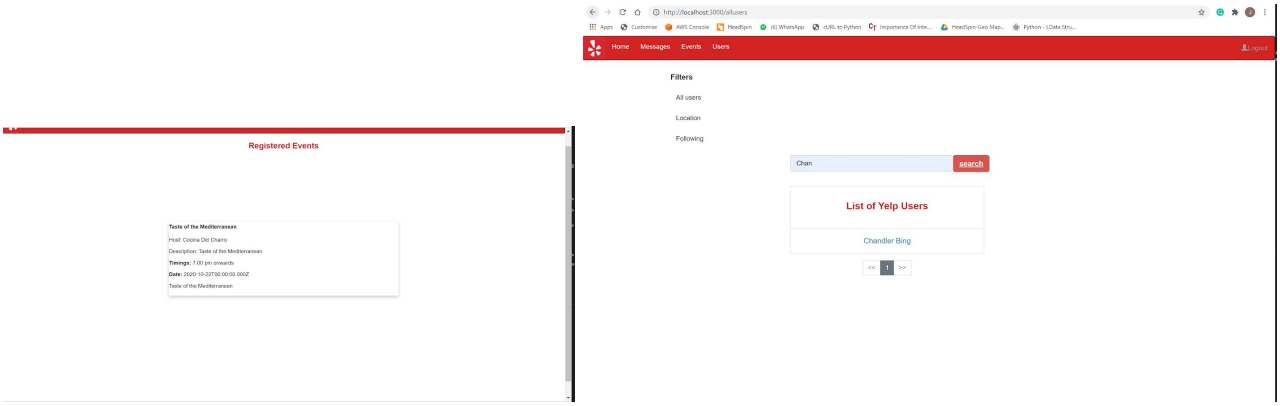
8. Users Tab

The user should be able to see all the yelp users. Pagination is implemented

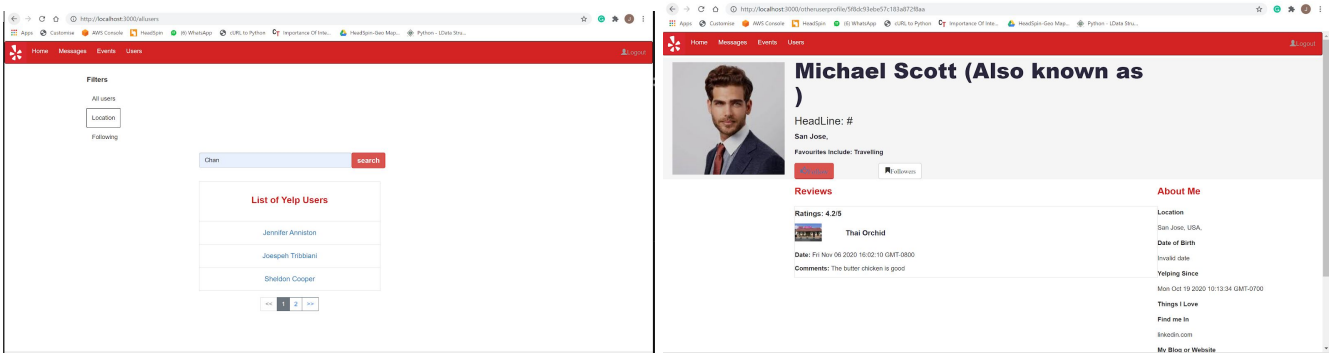
Redux stores the state of all users



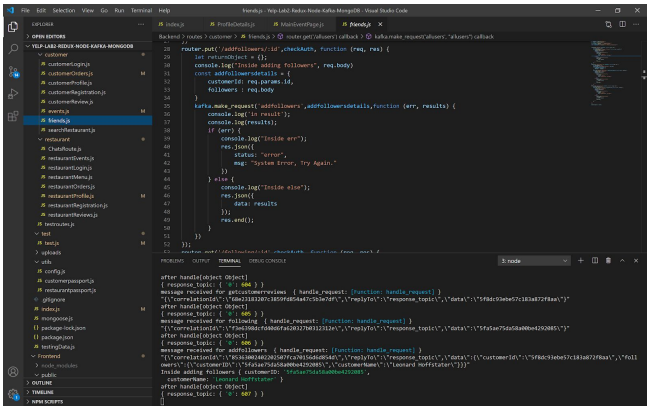
The user should be able to search for another user using his first name or nickname. (Here we are searching for Chandler Bing using Chan

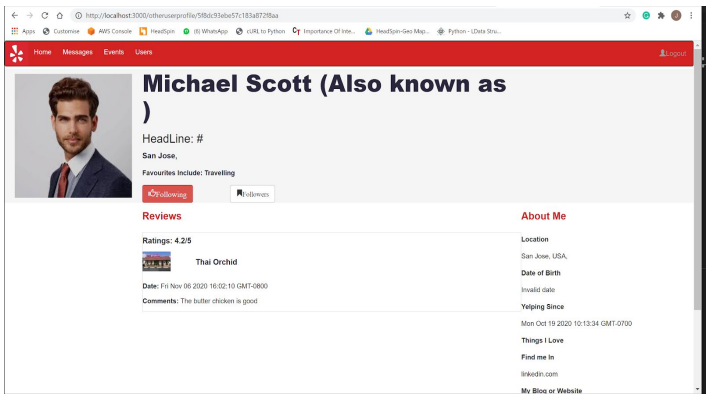


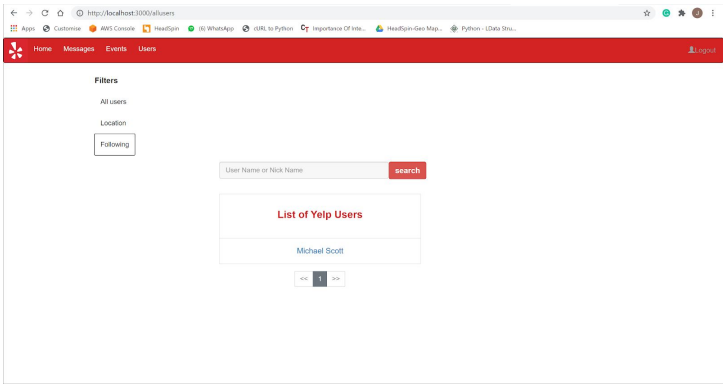
The results can be filtered using location
On clicking the user's name, the customer profile page should render



The user can follow another user and can filter the user results based on following



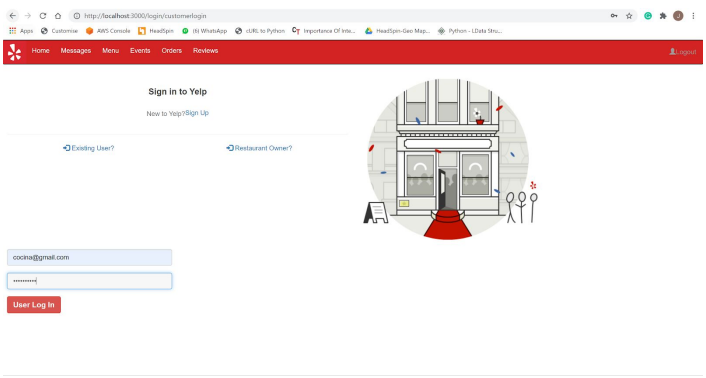


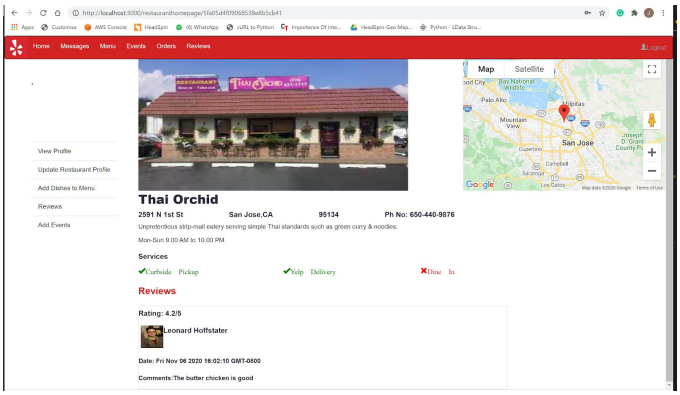


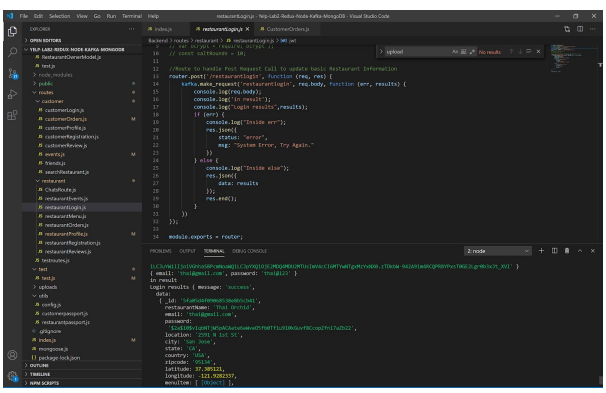
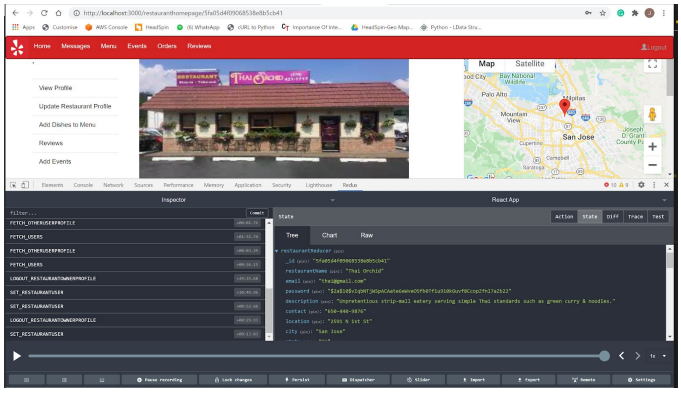
Restaurant Role:

1. Restaurant Login, Profile and Update Profile Details:

The restaurant owner registers using the restaurant name, email, password and location

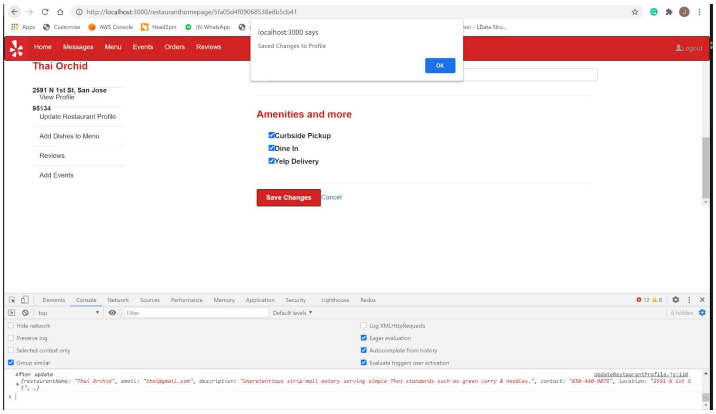
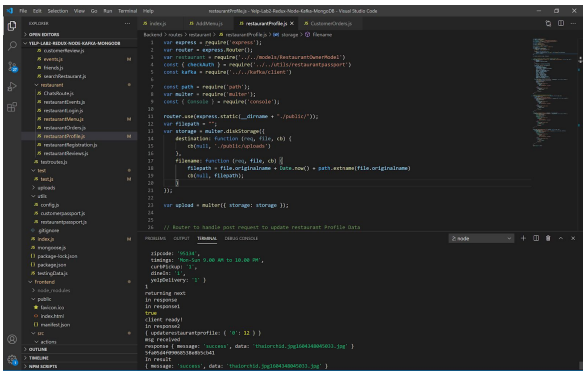




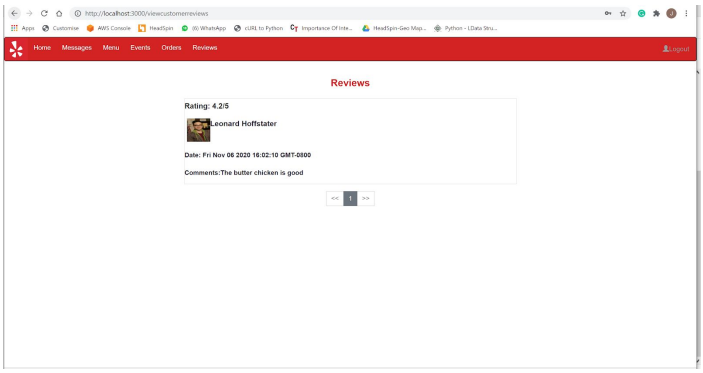
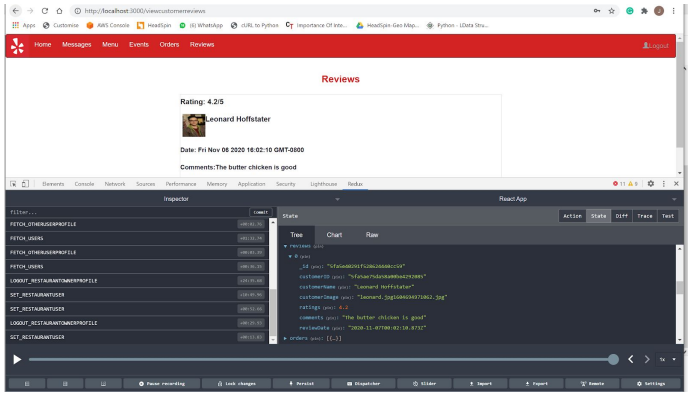
2. Update Restaurant Profile

The restaurant owner can update profile details such as basic details, restaurant pics, contact information etc

3. View Reviews

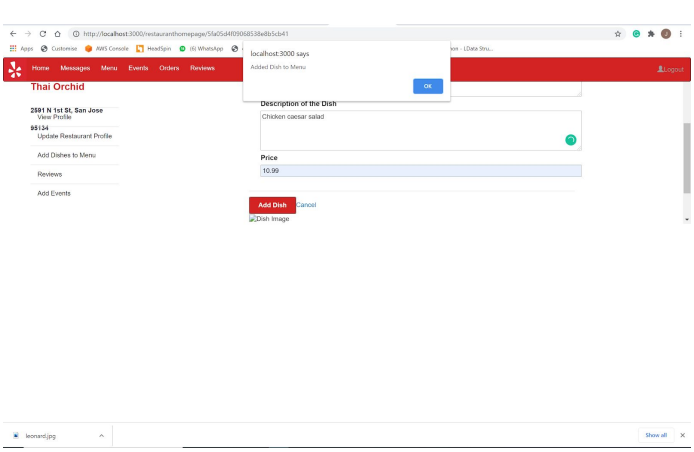
The restaurant owner should be able to view reviews given by all the users.
Pagination is implemented.
The reviews are fetched from the redux store

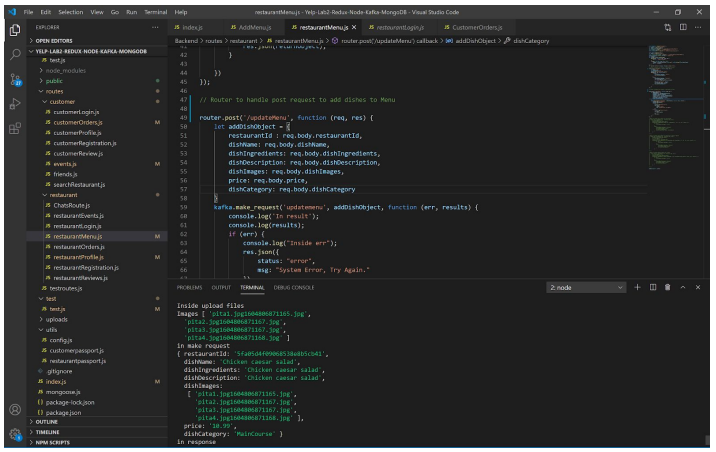



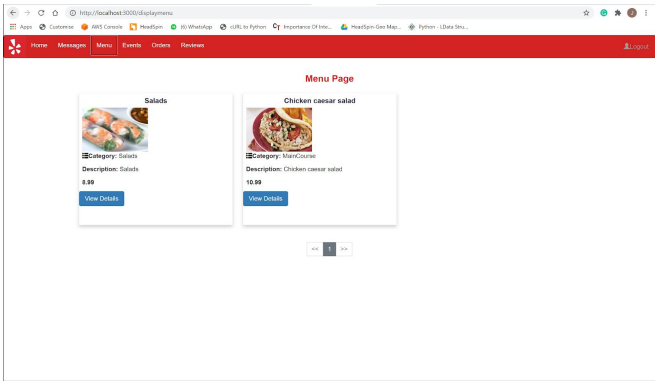
4. Add Dishes to the Menu

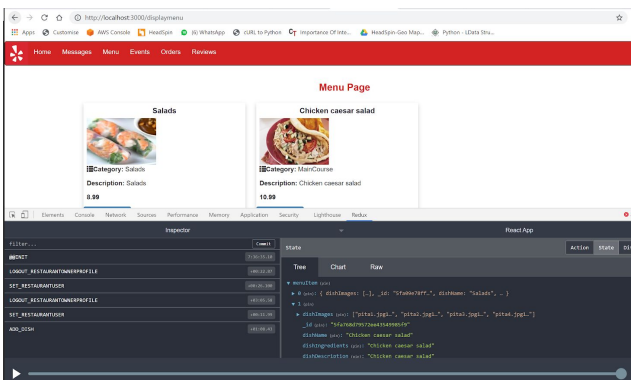
The owner can add dishes to the Menu, the added dishes can then be seen in the menu tab.

Pagination is implemented



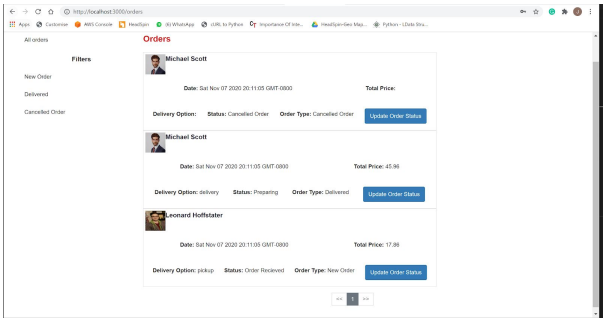


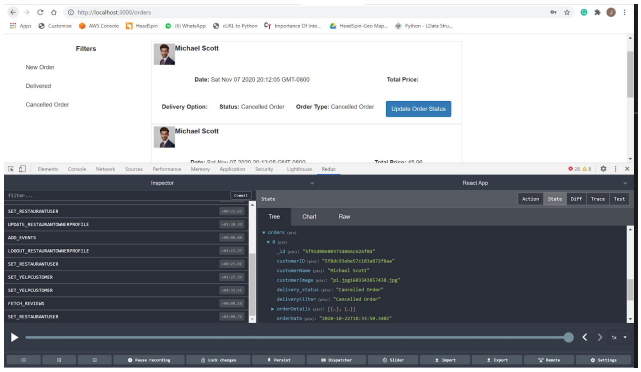


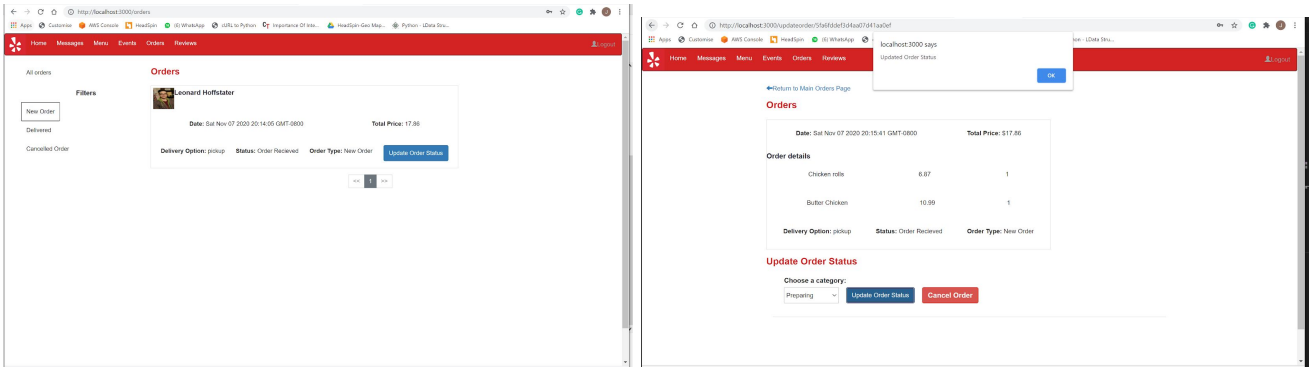


5. Orders Tab

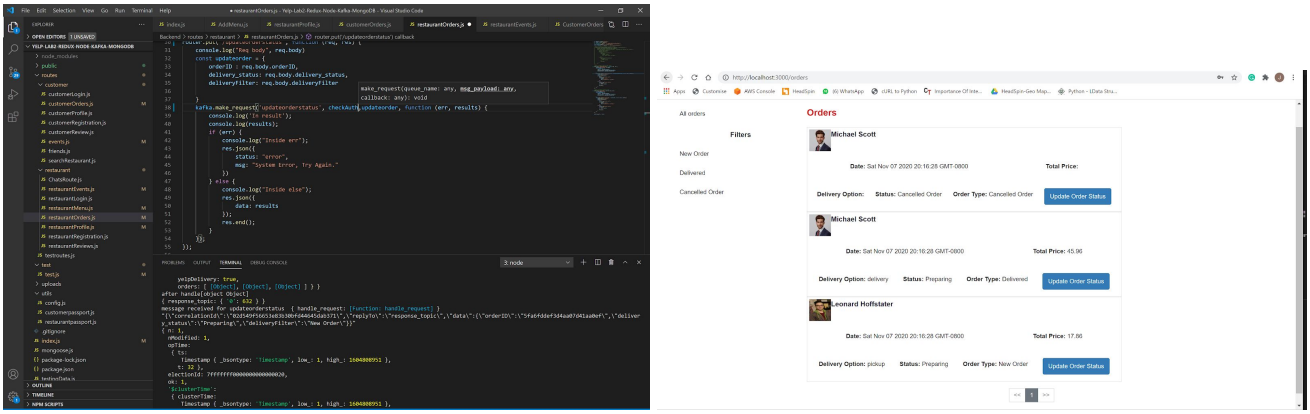
The restaurant owner is able to view all orders by customers. The order details are stored in redux. Pagination is implemented





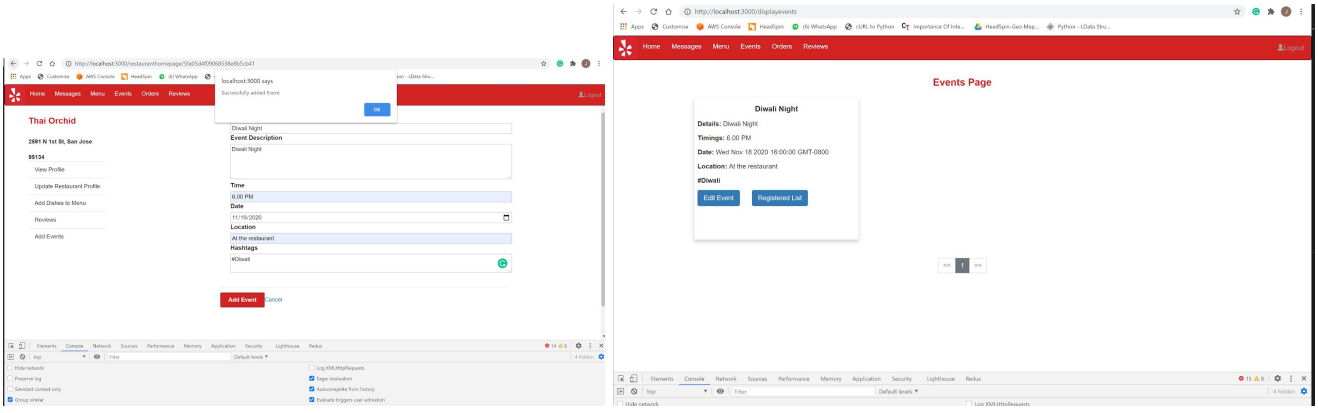


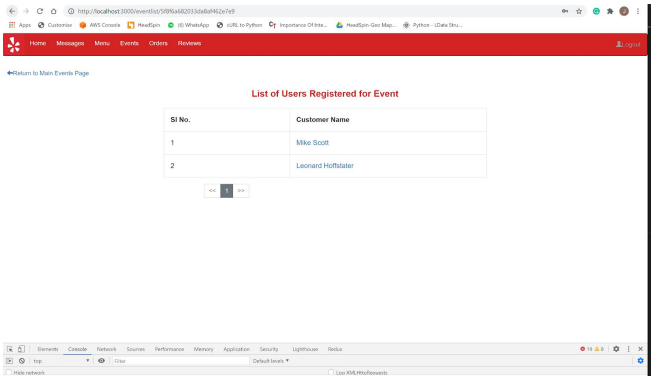
The orders can be filtered based on New order, delivered order or cancelled order.
Order status can be updated by the restaurant owner

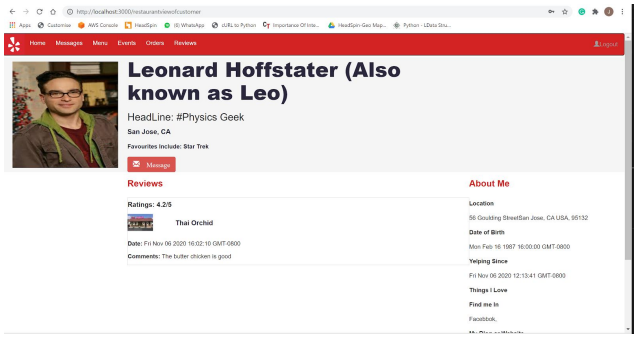


6. Events Tab

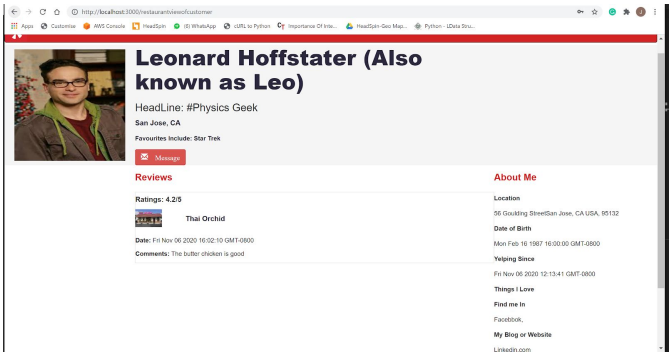
The restaurant owner should be able to post events. All the events posted by the owner can be viewed in the restaurant tab. The owner can view the registered list for each event and only clicking on the registered user, it should lead him to the customer page

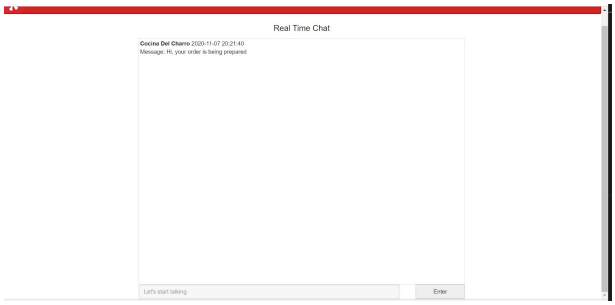


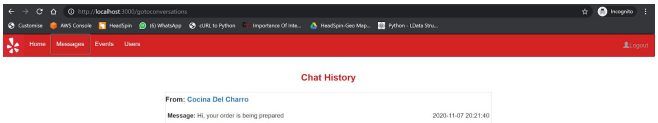


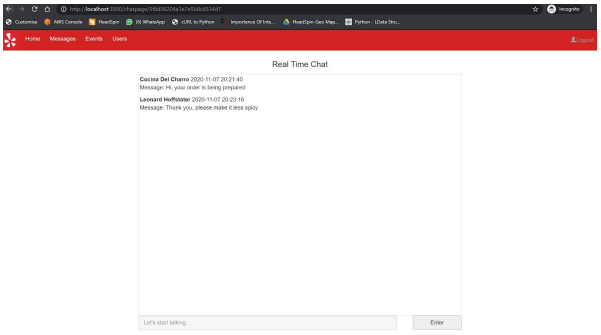


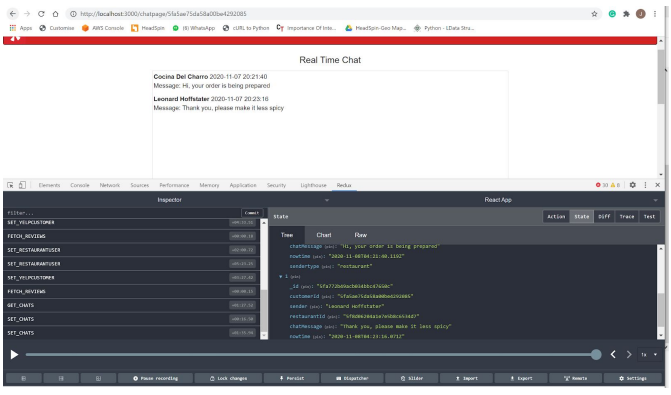
7. Messages Tab

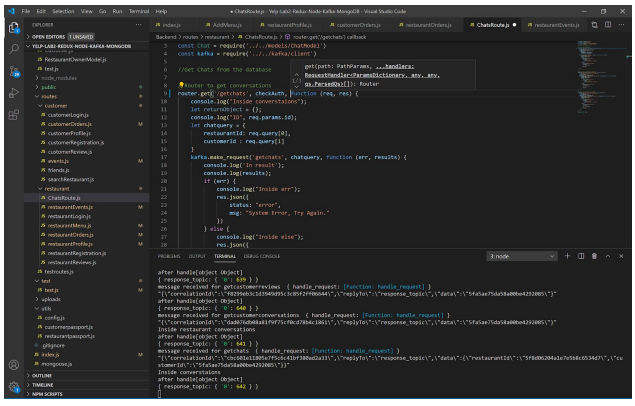








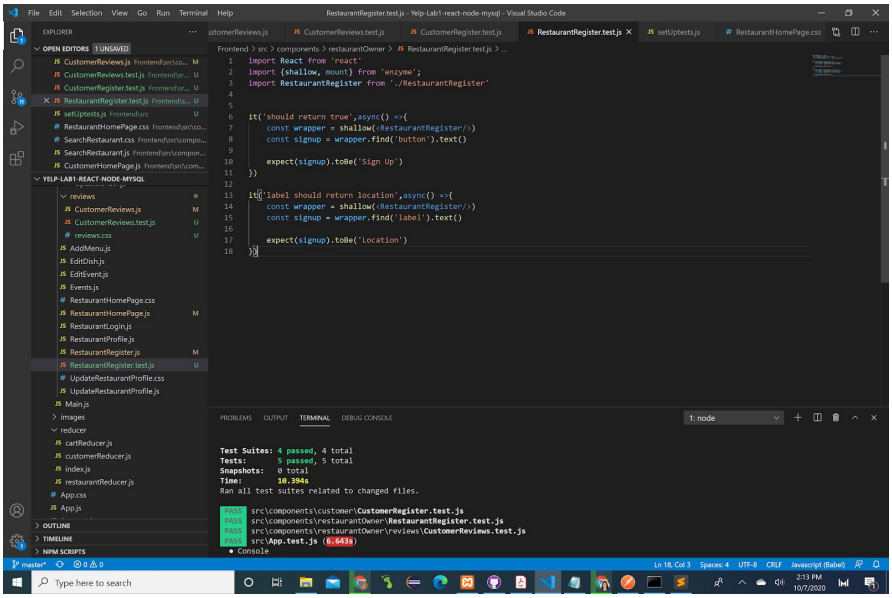




4. **Testing**

1. **Enzyme testing:**

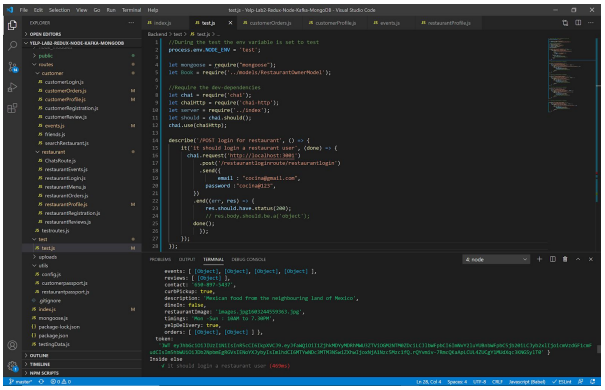
Enzyme testing was done for 3 views namely customer registration, restaurant registrations and reviews from customers

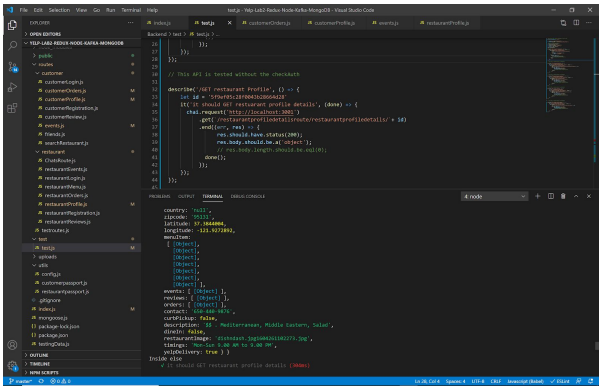


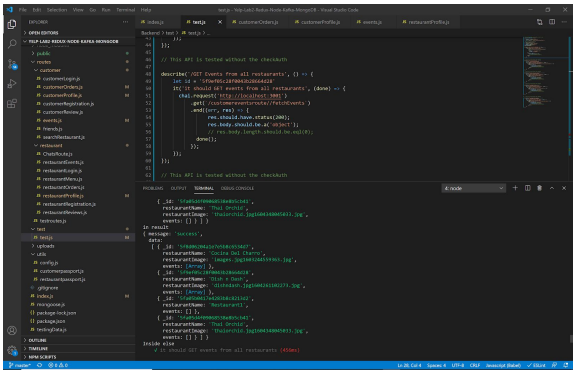
2. **Mocha Chai Test**

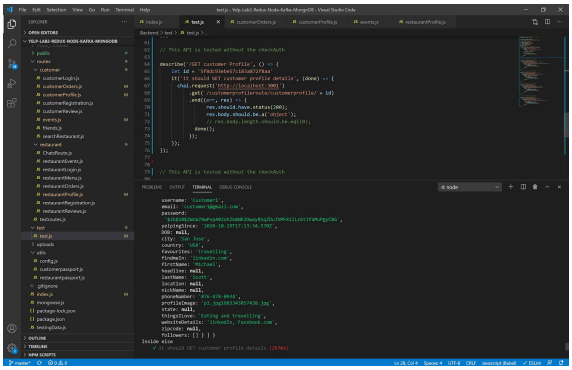
Mocha chai testing was implemented for 5 random API's

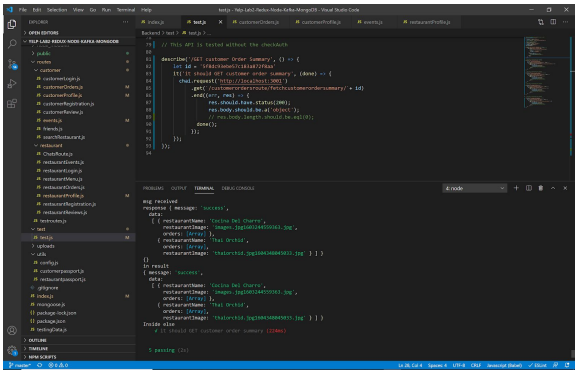
1. Restaurant Login
2. Fetch restaurant profile for a particular customer
3. Fetch all restaurant events
4. Fetch customer profile
5. Fetch order summary for a particular restaurant











3. Performance testing using JMeter

Jmeter is used to test the restaurant login route

1. MongoDB JMeter testing was done using connection pooling and without connection pooling. For connection pooling the pool size was set to 100.
2. From the below results, the response time increases proportionately when the number of concurrent users increases.
3. When the pool size is equal to or lower than the concurrent users, the average response time is much faster. As the number of users increases about the pooling size, the response time increases exponentially
4. Time taken for requests to be served on a non pooled connection is more since it can serve only one server request at a time, this causes a performance bottleneck as the response time increases

API	**Users**	**With connection Pooling(ms)**	**Without Connection Pooling(ms)**
http://localhost:3001/restaurantloginroute/restaurantlogin	100	795	1835
	200	1986	2987

	300	3996	4864
	400	4844	5949
	500	6241	8009

With Connection Pooling

100 users

200 users

300 users

400 users

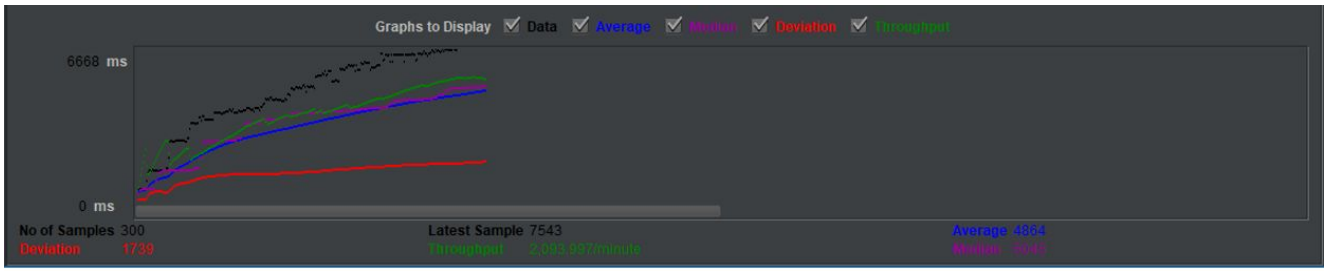
500 users

Without using Connection Pooling

100 users

200 users

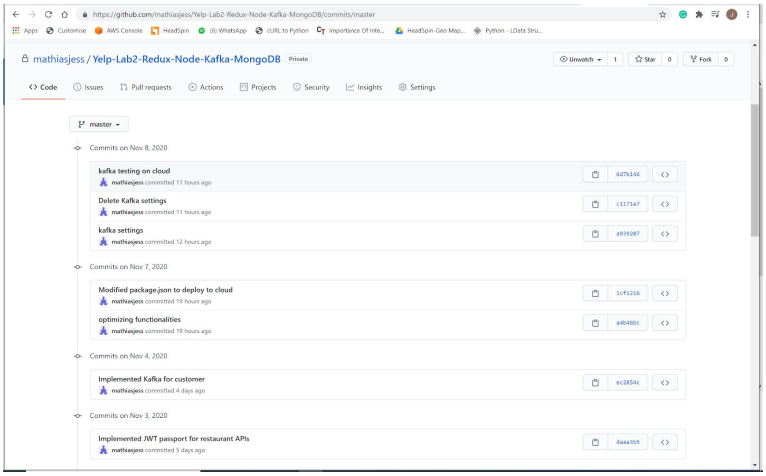
300 users

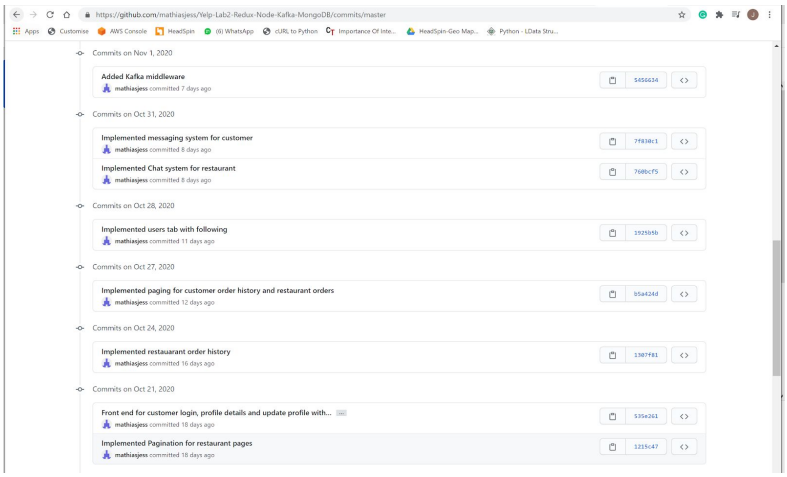


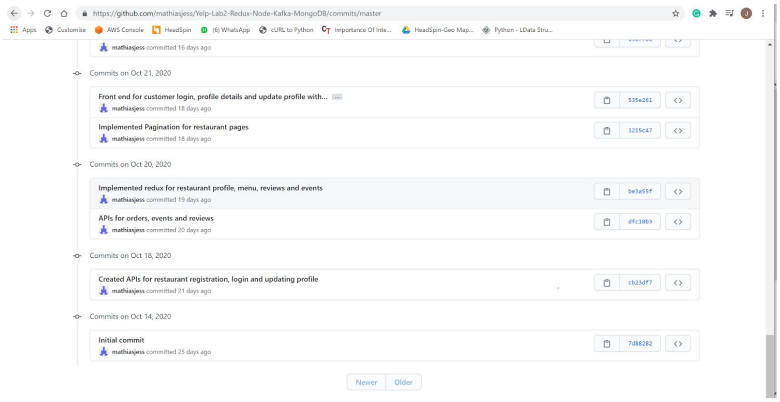
400 users

500 users

5) Git Commit History







6. Answer to the questions

1. Comparison of passport authentication with Lab1 authentication process.

In the first lab for the authentication process, bcrypt was used to encrypt the password and the token was stored in the localStorage. This method did not provide security to the API calls. With JWT, a token is

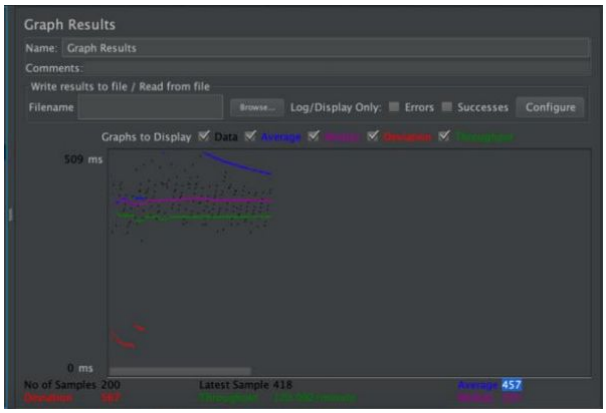
created during the login for a restaurant or customer. This token can then be used to authenticate the roots and make any requests. The middleware created using passport can be reused by multiple API's which helps to increase code reusability and decreases code complexity

2. Performance comparison of with and without Kafka.

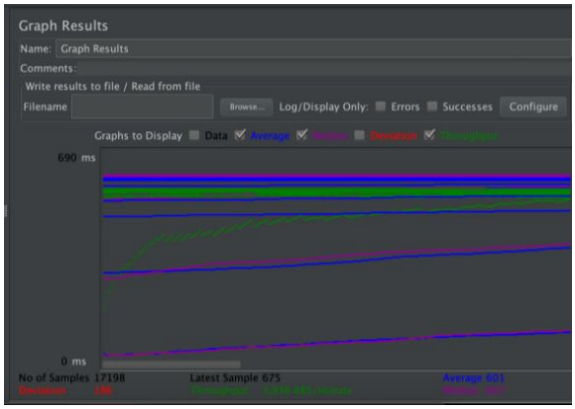
One API was used in order to compare the performance of response time with and without Kafka messaging system. The application was load tested with 200 users for 10 minutes. It was found that with kafka the average response time was < 0.4 sec whereas without Kafka the average response time was greater at >0.6 sec

The Kafka queues were able to deliver the backend messages, as shown, and also acted as a store buffer to escape bottlenecks from the database. Not only are responses enhanced, but Kafka will help the application scale in a distributed manner. As the available queue will pick up the message to prevent bottlenecks from being processed.

With Kafka



Without Kafka



3. If given an option to implement MySQL and MongoDB both in your application, specify which data of the applications will you store in MongoDB and MySQL respectively

MySQL which is a relational database is highly reliable when it comes to maintaining consistency in data. MongoDB on the other hand gradually becomes consistent. With this as consideration, critical data which is required for almost all API requests and needs consistency should be stored in the MySQL database.

In the yelp application, in the case of a customer, we need the custom ID, email and password across APIs to authenticate and fetch consequent requests. In the case of a restaurant we need the restaurant ID, name, email, password and location to make subsequent API requests. Hence these data fields with high dependency on them should be stored in the MySQL so that retrieval is quick and consistent

MongoDB works best when there is a need to store non structured data. Since the data is non relational, it fetches data faster. For example in the customer role, customer details can be stored in the mongoDB, orders, registered events, and messages can be stored in the MongoDB

In this application, MongoDB played a crucial role for me in creating the restaurant schema. WIth MongoDB it was easier to structure the events, menu items, reviews and orders for a particular restaurant under one schema. This made it easier to pull the data from MongoDB once and store it in the redux state. With one call, all the details for the restaurant can be fetched

In case of messaging system, it is very useful to store the chats in the MongoDB since the data can be non structured ie, array of text, images and videos can be stored in MongoDB

Another instance is that auto sharding will benefit from the inclusion of items in MongoDB for Items set. This will also suggest that sets of objects are essentially compatible (which could be a tradeoff considering this as non-critical data from the rest of the request)

Note:

Added TA as contributor