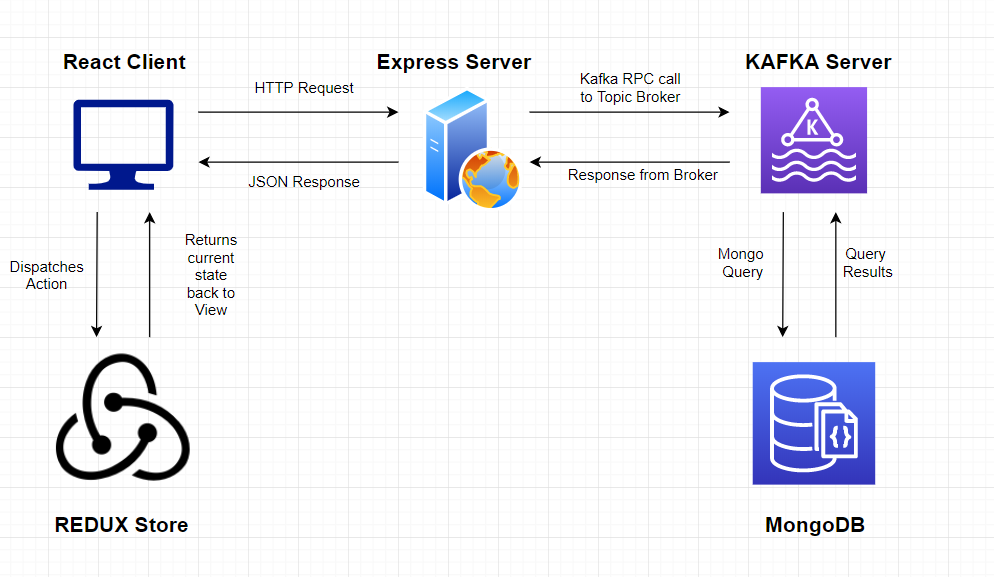
**GRUBHUB APPLICATION**

**System Design**



**React Client**: UI for the application running on client’s web browser.

**Redux Store**: State management in the front-end.

**Express Server:** Backend server responsible for handling HTTP requests from client. Also serves as Kafka producer and sends out messages to Kafka server based on topics.

**Kafka Server**: Maintains a messaging queue. Connects to MongoDB. Handles messages with appropriate DB calls and returns the results back to the Express Backend.

**MongoDB**: NoSQL database to store all records/documents.

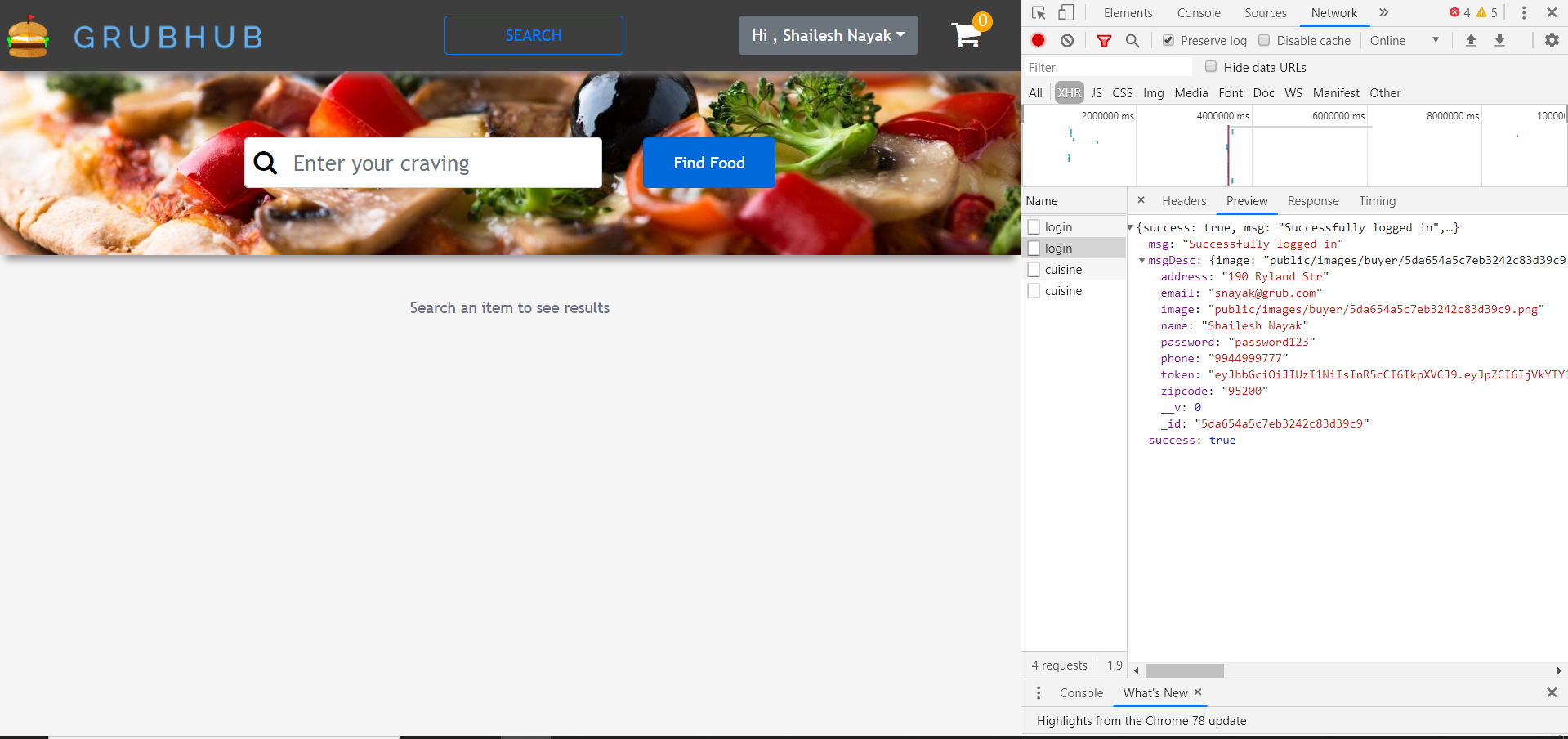
**Purpose of the system :**

To build a prototype of GrubHub food ordering application.

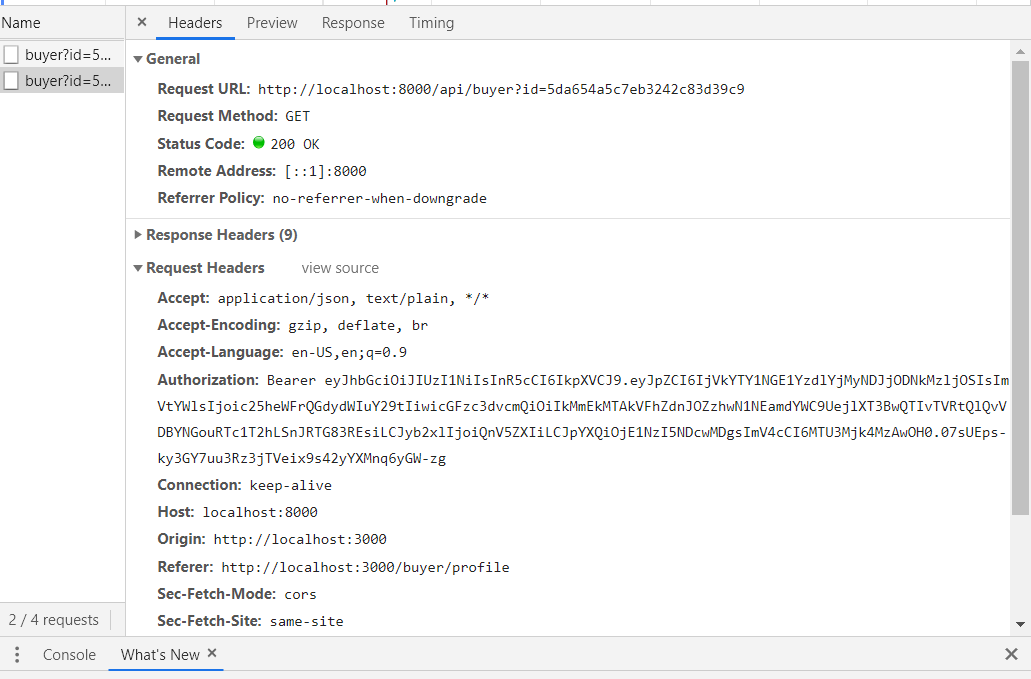
Customers can order food from restaurants and Restaurant owners can sell items using a section wise divided menu and manage orders they receive from customers.

**Client & Server screenshots *(API response is shown in Networks tab of Chrome Devtools)***

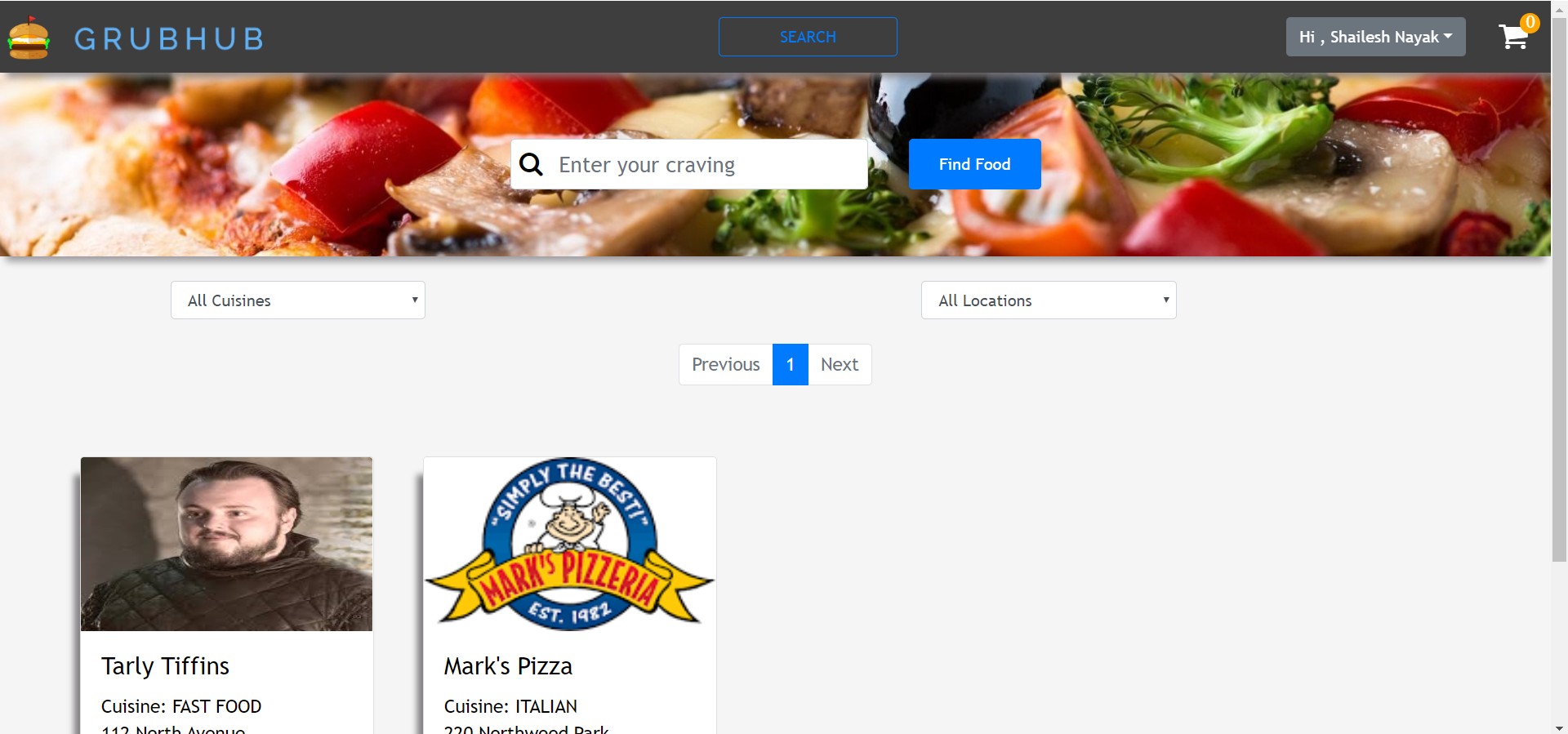
1. HOME PAGE FOR CUSTOMER AFTER LOGIN (JWT TOKEN)



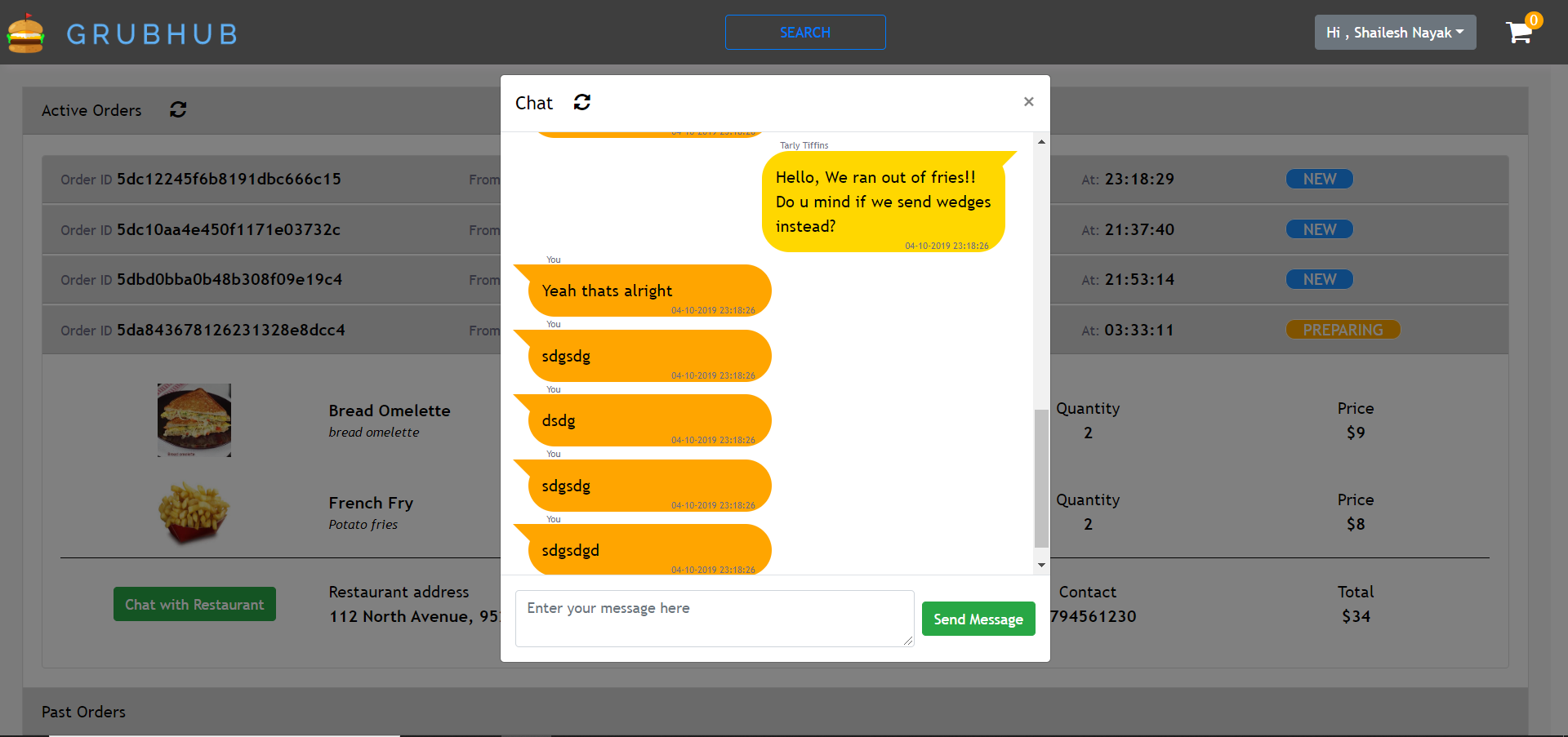
1. PROFILE PAGE (REQUEST MADE USING JWT TOKENS)



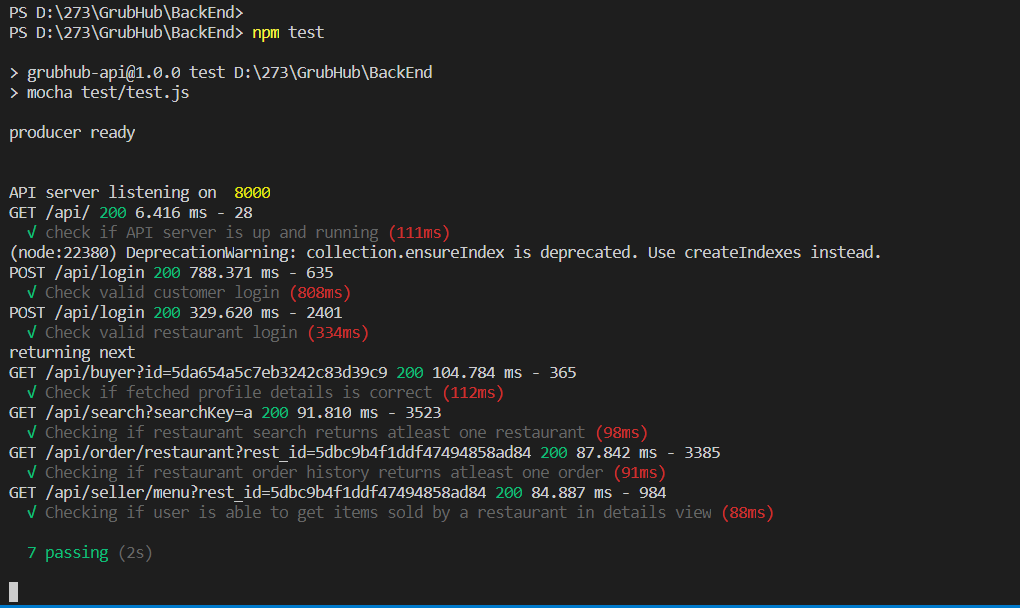
1. SEARCH PAGE WITH RESULTS (PAGINATION ADDED)



1. CHAT FEATURE



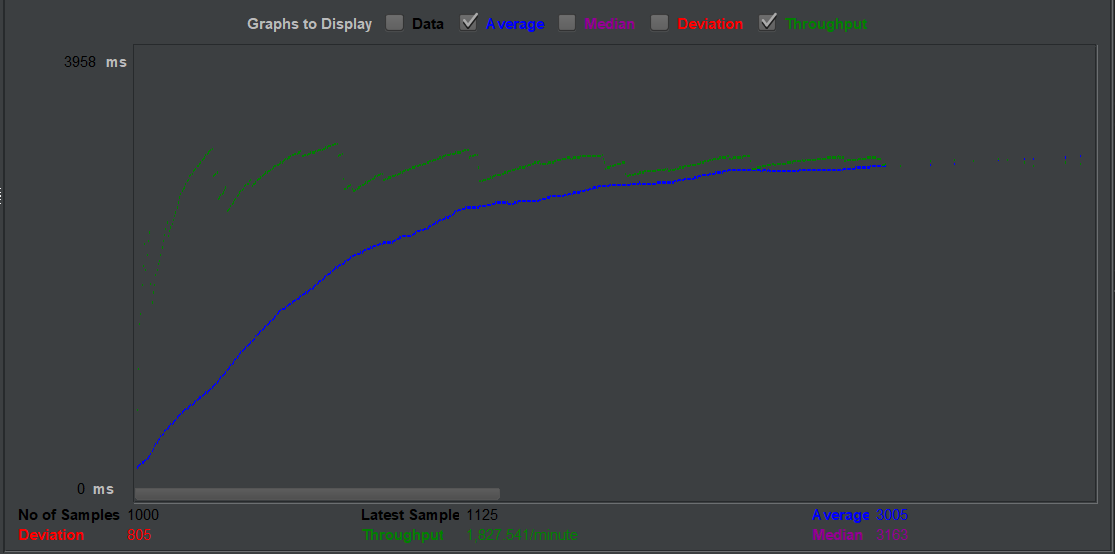
**MOCHA TEST RESULTS**

****

**PERFORMANCE TEST – JMETER RESULTS**

**\*All tests performed on SEARCH CALL - ‘api/search’\***

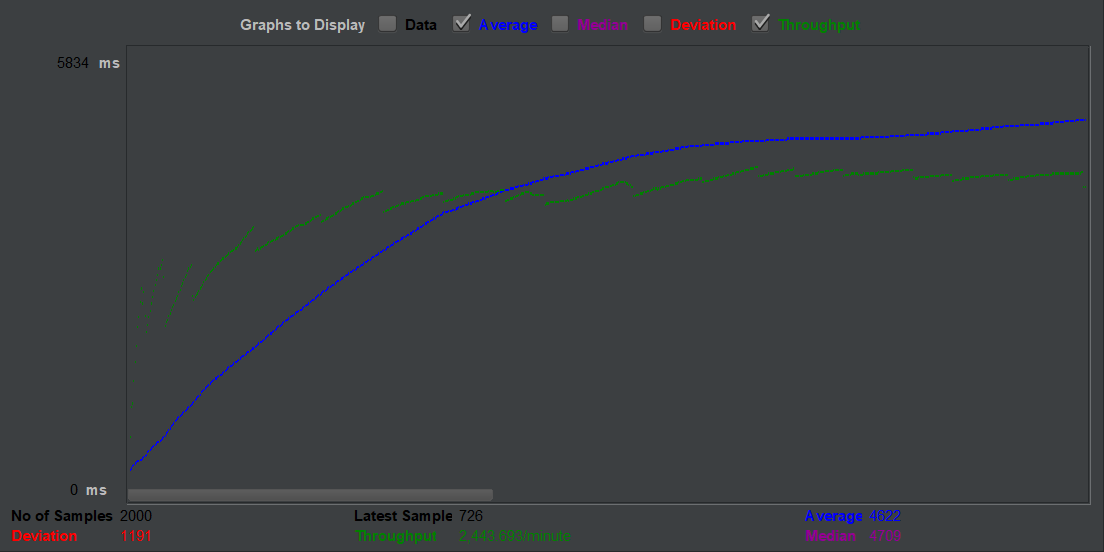
1. **Concurrent Users – 100 *(each making 50 calls each)***
2. **Without connection pooling (Average = 3005ms)**

****

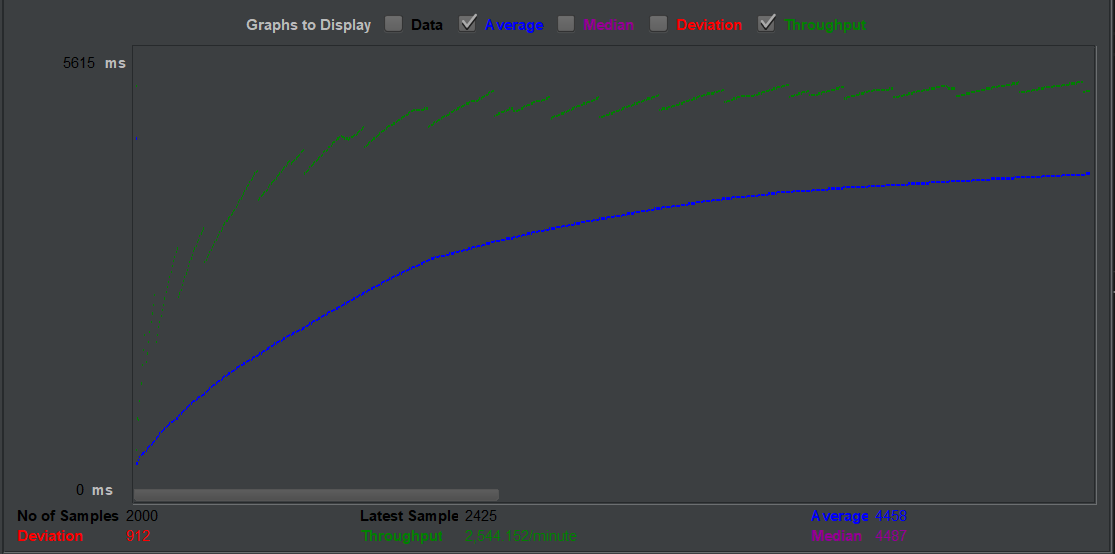
1. **With connection pooling (Average = 2268ms)**

****

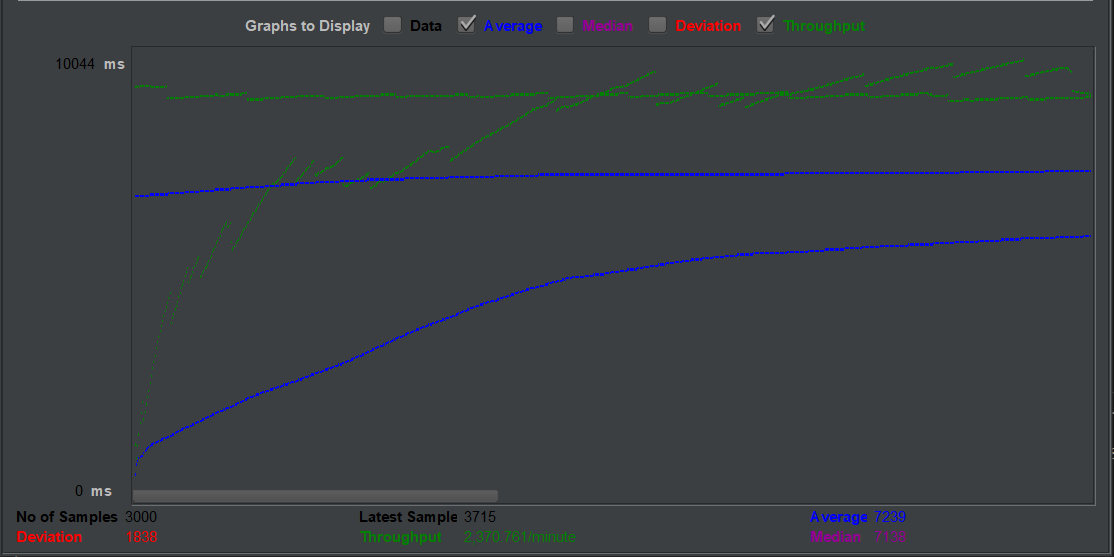
1. **Concurrent Users – 200 *(each making 50 calls each)***
2. **Without connection pooling (Average = 4622 ms)**

****

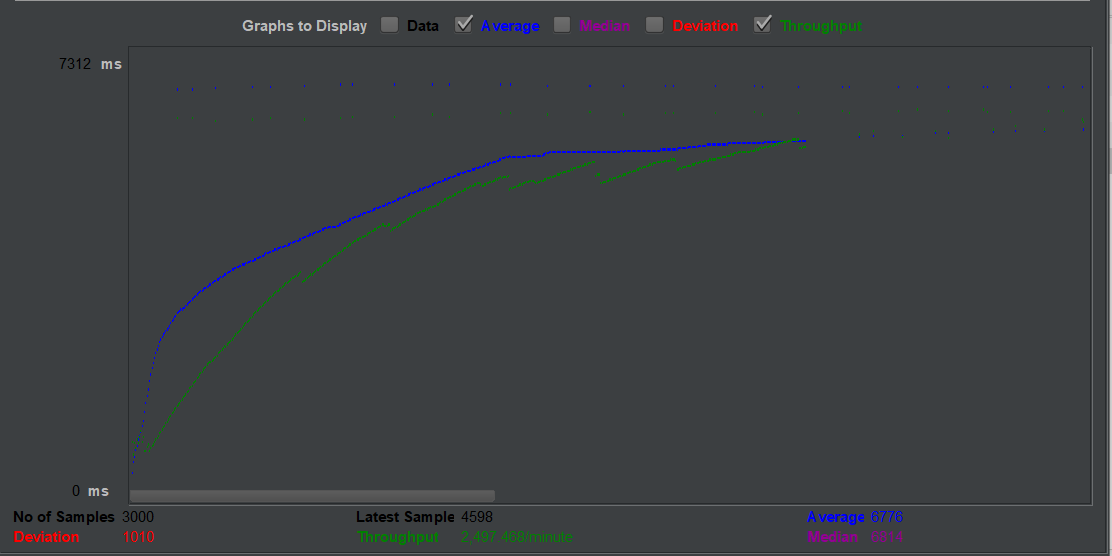
1. **With connection pooling (Average = 4458 ms)**

****

1. **Concurrent Users – 300 *(each making 50 calls each)***
2. **Without connection pooling (Average = 7239 ms)**

****

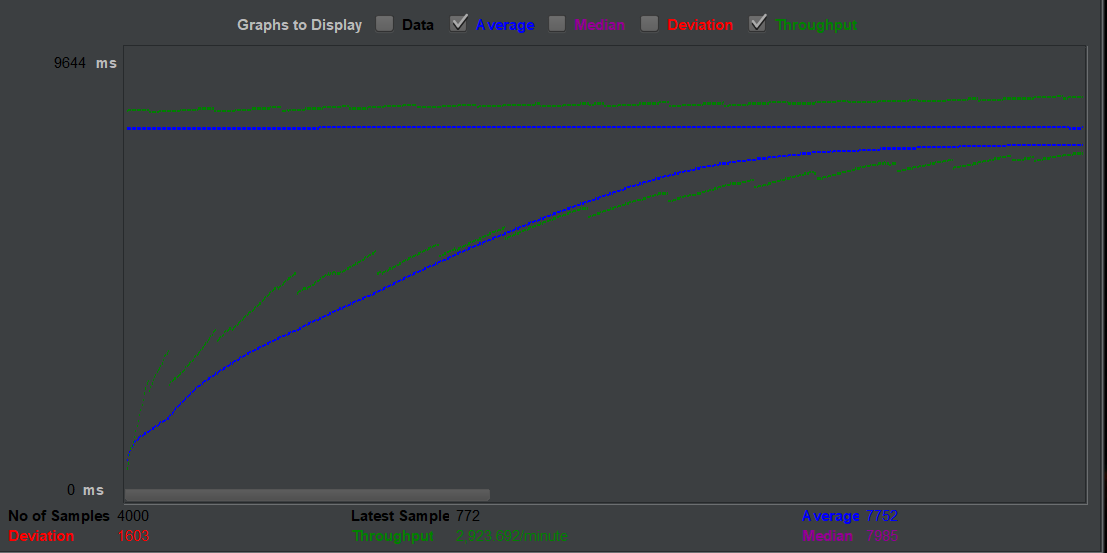
1. **With connection pooling (Average = 6776 ms)**

****

1. **Concurrent Users – 400 *(each making 50 calls each)***
2. **Without connection pooling (Average = 8091 ms)**

****

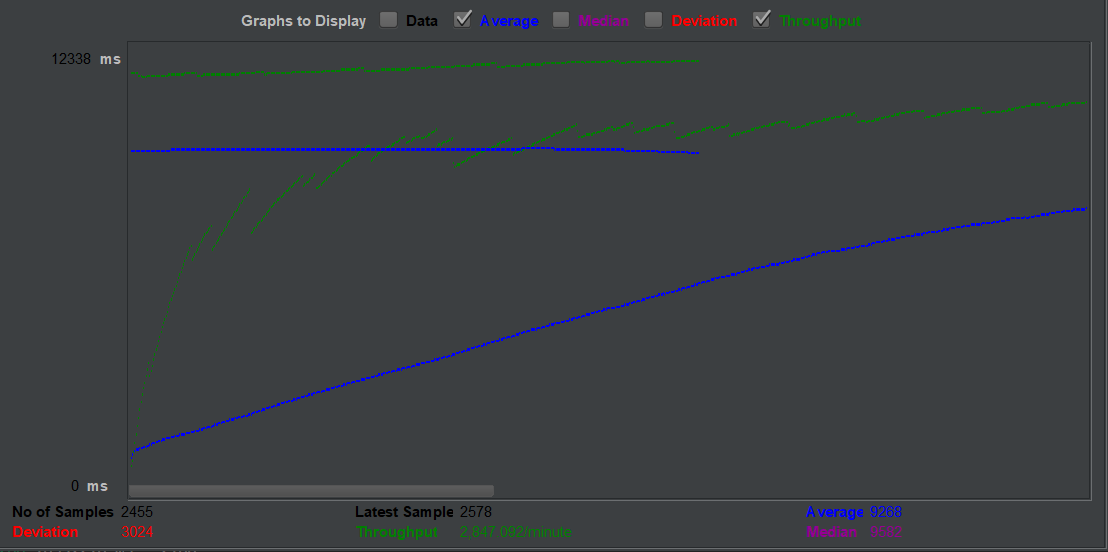
1. **With connection pooling (Average = 7752 ms)**

****

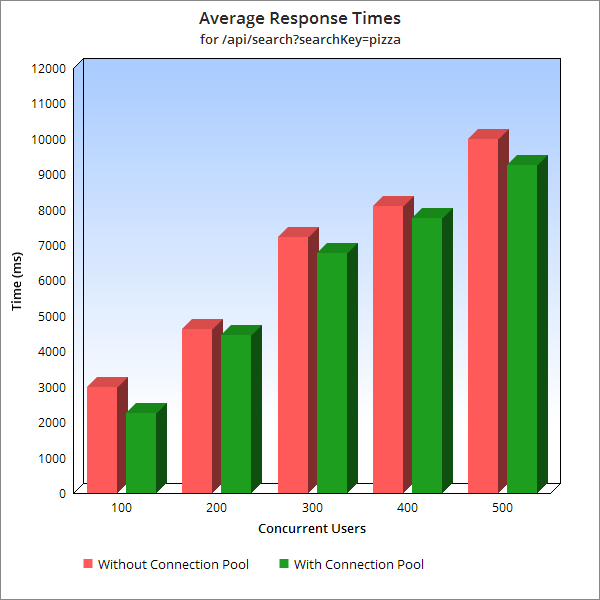
1. **Concurrent Users – 500 *(each making 50 calls each)***
2. **Without connection pooling (Average = 10002 ms)**

****

1. **With connection pooling (Average = 9268 ms)**

****

**Analysis:**

****

**From the observations of the JMeter tests and the graph plotted against it’s results, we can clearly conclude that Connection Pooling has provided ~6% decrease in average response time for the search API call.**

**DEPLOYMENT**

**GRUBHUB application is deployed on AWS EC2 AMI (Ubuntu instance)**

1. **Frontend and backend are running on different ports on the same instance.**
2. **BackEnd and Kafka Backend are running on different ports on the same instance.**
3. **Mongo DB is hosted on an Mongo Atlas Instance**

**Public DNS of FrontEnd :**  <http://ec2-18-217-100-107.us-east-2.compute.amazonaws.com:3000/login>

**Public DNS of BackEnd:** ec2-3-19-14-1.us-east-2.compute.amazonaws.com

**Repo Link :** <https://github.com/shailesh-nyk/Lab2-014529151>