**ETSY CLONE**

Youtube link: <https://www.youtube.com/watch?v=PhYHIPiKI3o>

**Introduction**

Full stack application of Etsy clone. Etsy is an ecommerce website.

Roles involved:

1. Buyer (Customer), Search for any product in the dashboard, add to favourites collection, add multiple items to cart, place order, my order history.
2. Seller (Create a shop and become a seller), sellers can create a unique shop name, add new item to shop, edit old items without impacting previous orders.

Only sellers can edit their own shop, and items. Customers can view items from all shops.

**System Design**

Frontend layer: React, Redux

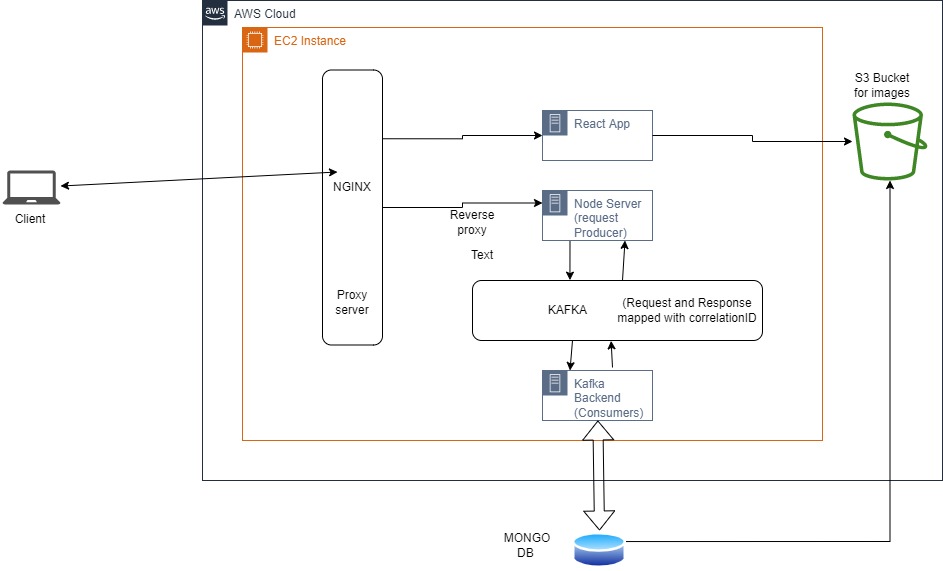
Server Side: Node JS

Database: MongoDB: Mongo Cloud for Database

Messaging Queue: Kafka

AWS cloud: EC2 Instance (ubuntu), S3

Nginx Web server.

****

Etsy clone application is deployed on to AWS cloud.

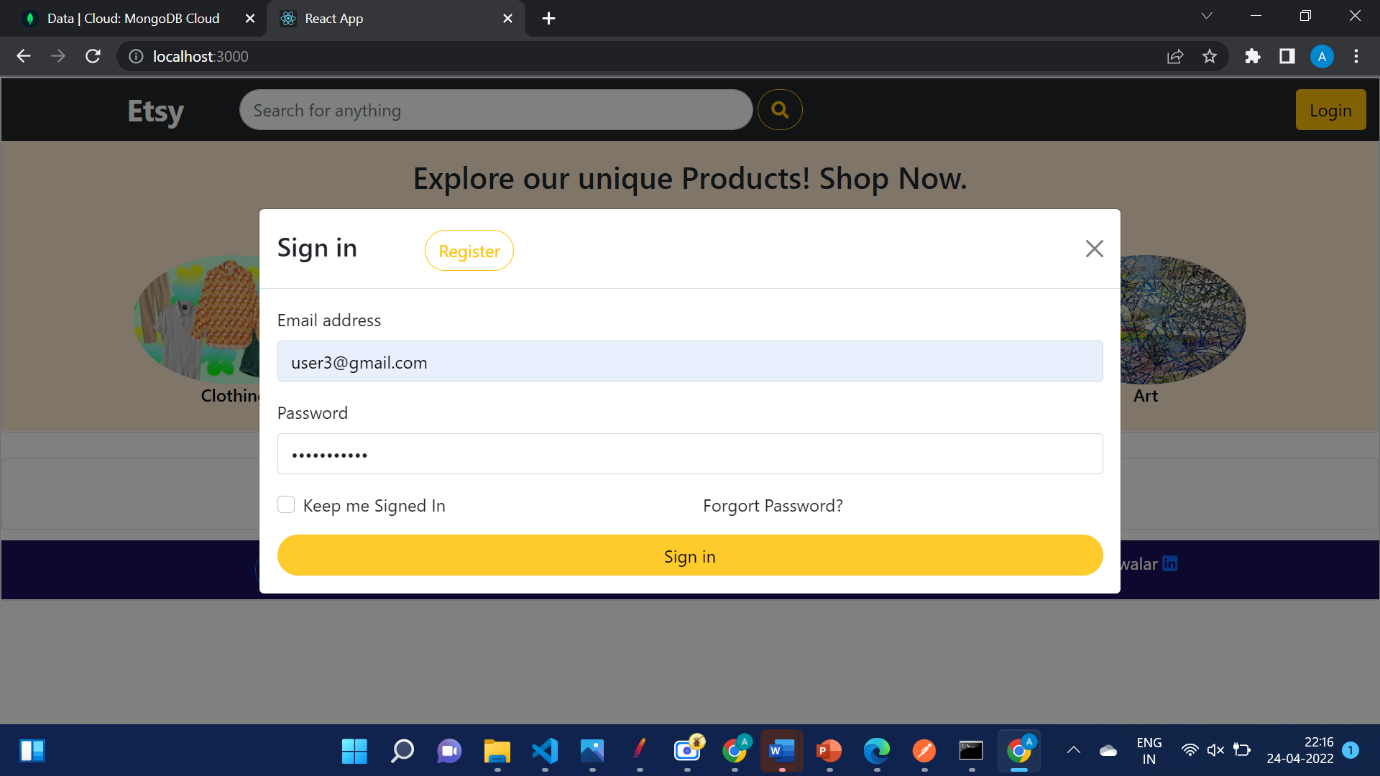
**Nginx** is used inside EC2 instance to serve the client request, Nginx is a open source web server that provides capabilities like reverse proxying, caching and load balancing. Here, Nginx servers the React App server through build folder, requests from 65.0.26.19/ are served from /index.html in react app, and requests /api/\* are forwarded to port 8585 (node server), therefore nginx acts as an intermediate server that intercepts client requests and forwards them to appropriate backend server, thus provides a layer of ABSTRACTION.

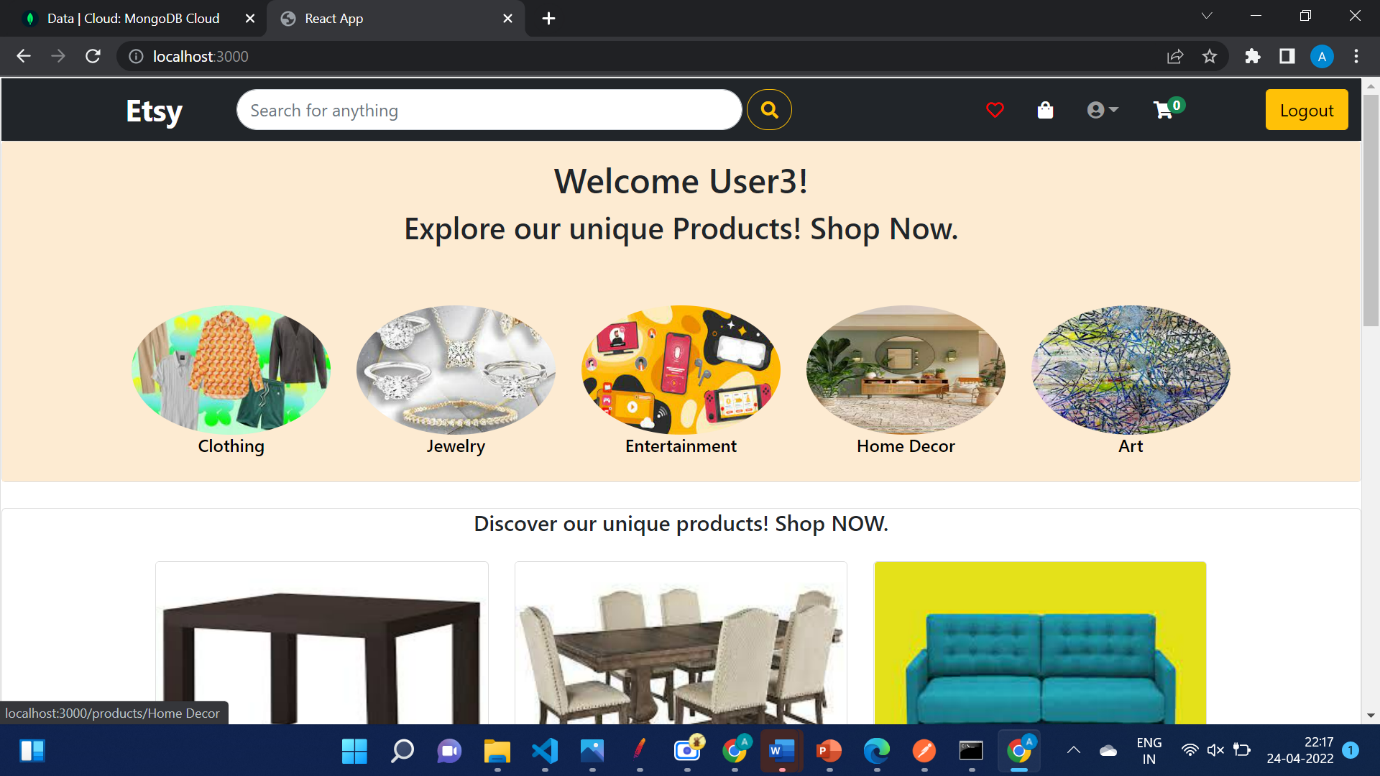
AWS **S3** is leveraged to upload client images from the application to a public bucket. The image URL is stored in the database.

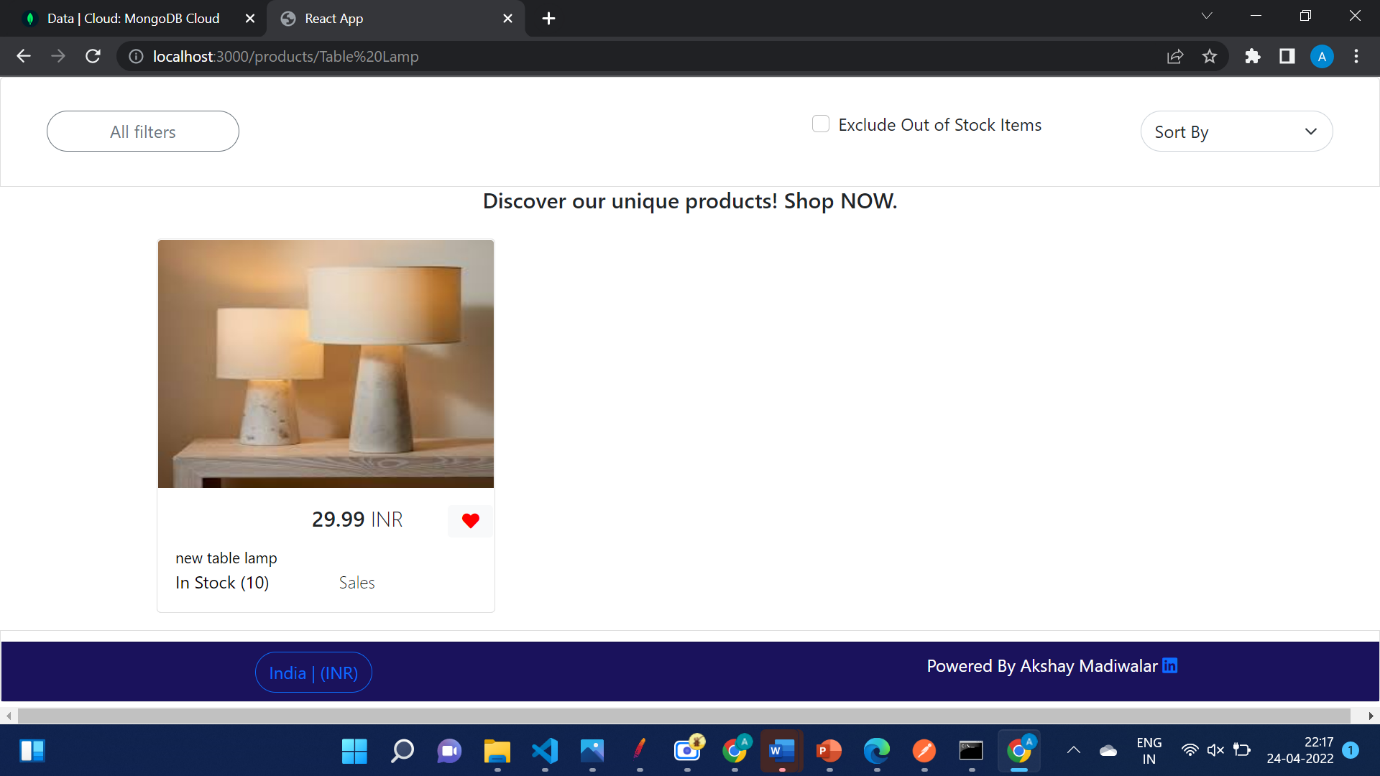
Mongo Atlas is used for the Mongo dB Database.

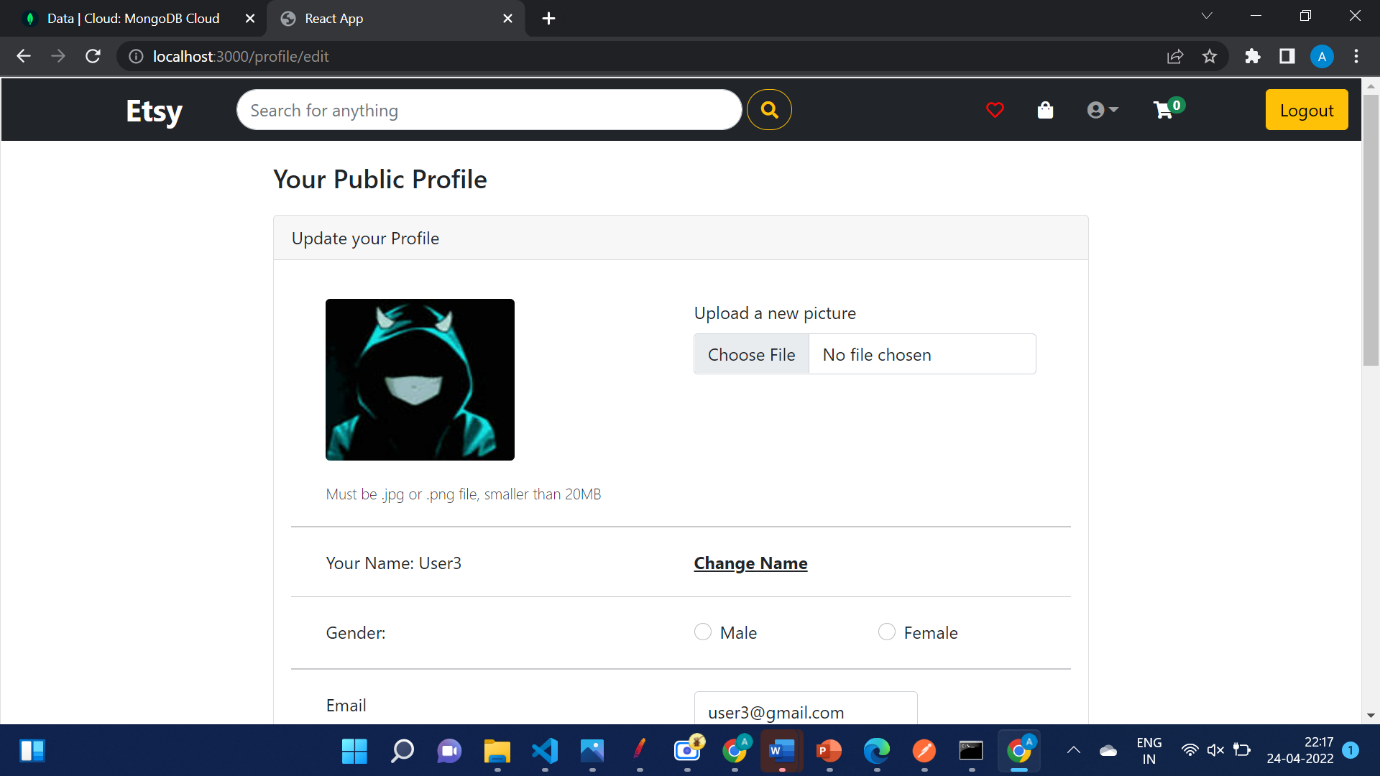
**Kafka** is used as messaging queue, for the request and response cycle. All the requests are sent to a kafka topic each request is attached with a unique correlation Id, the consumers in other service are subscribed to the topics, consumes the payload (requests) and handles the actual (talk to DB) and sends back the response attaching the same correlation ID which will be later used to match the request and response.

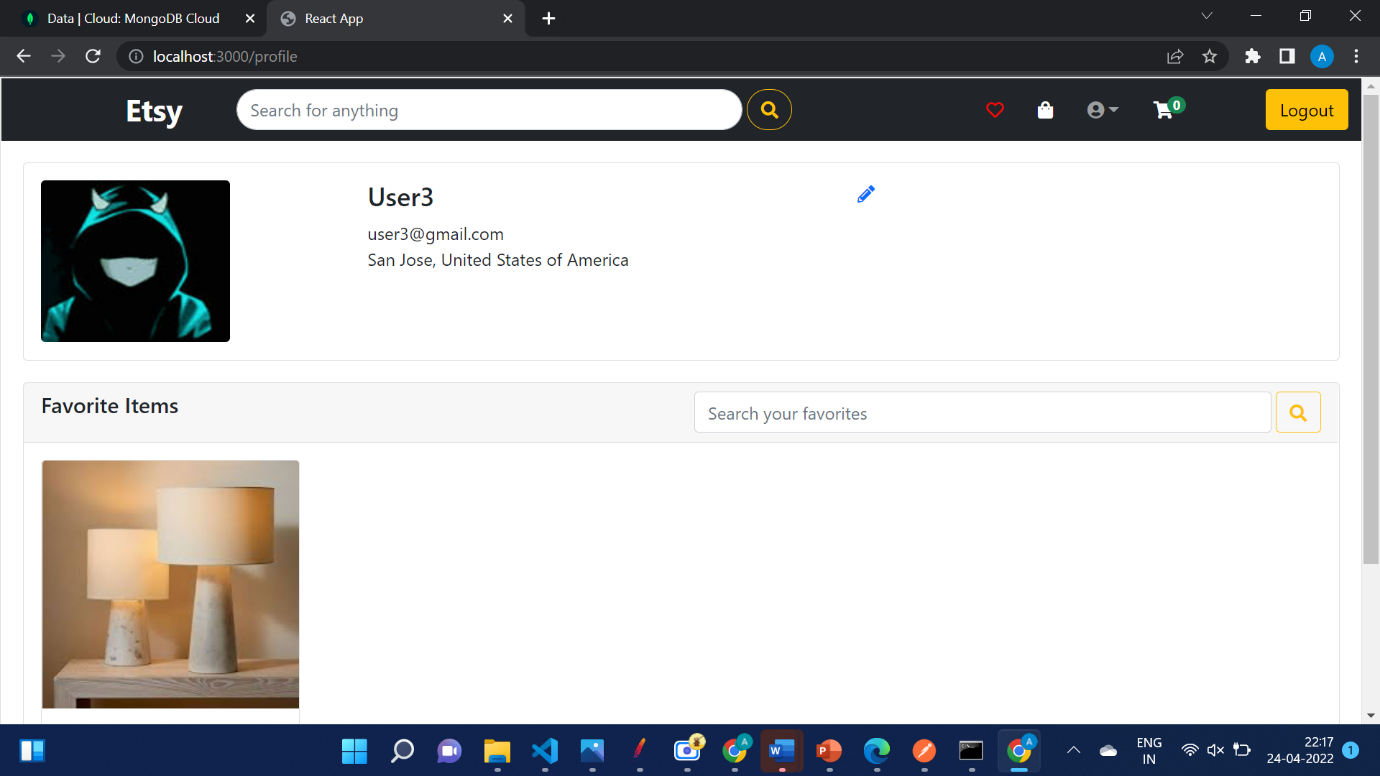
**Results**

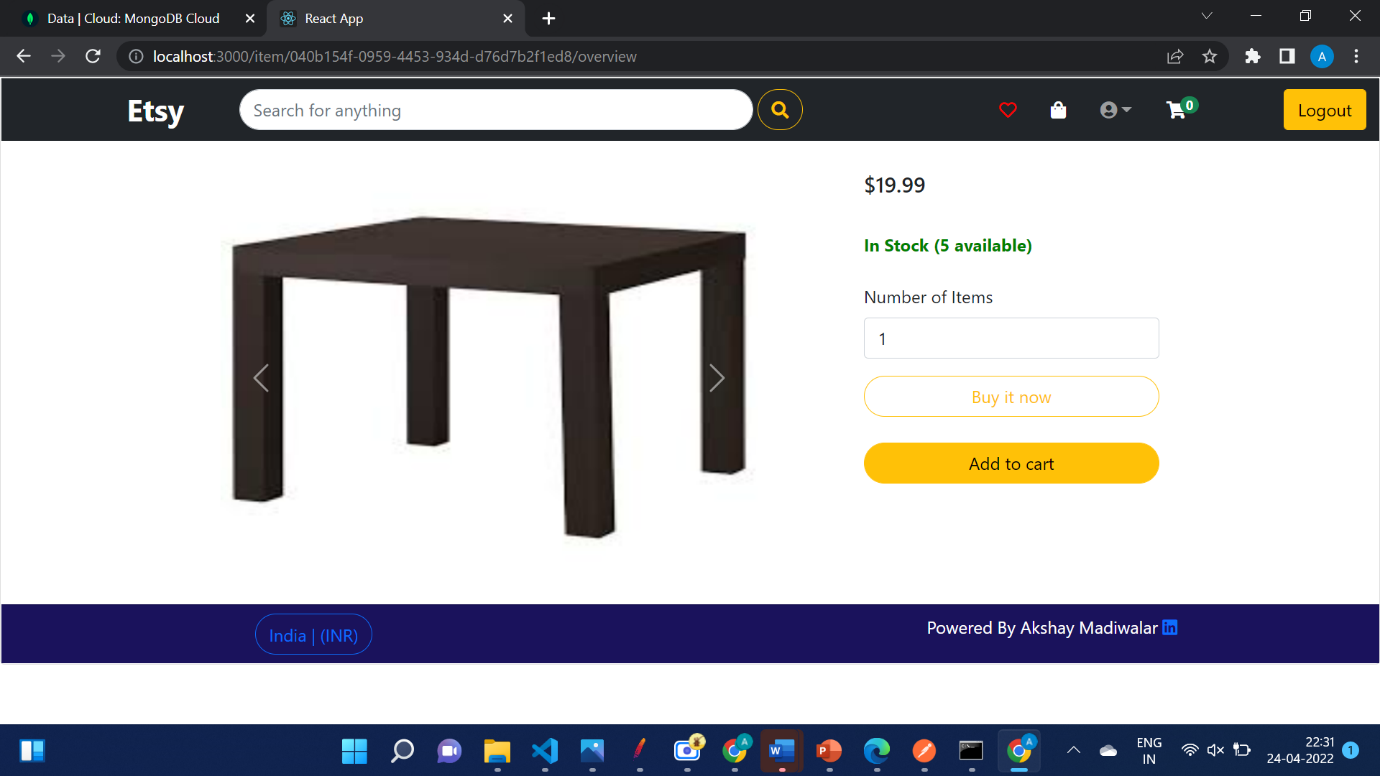
****

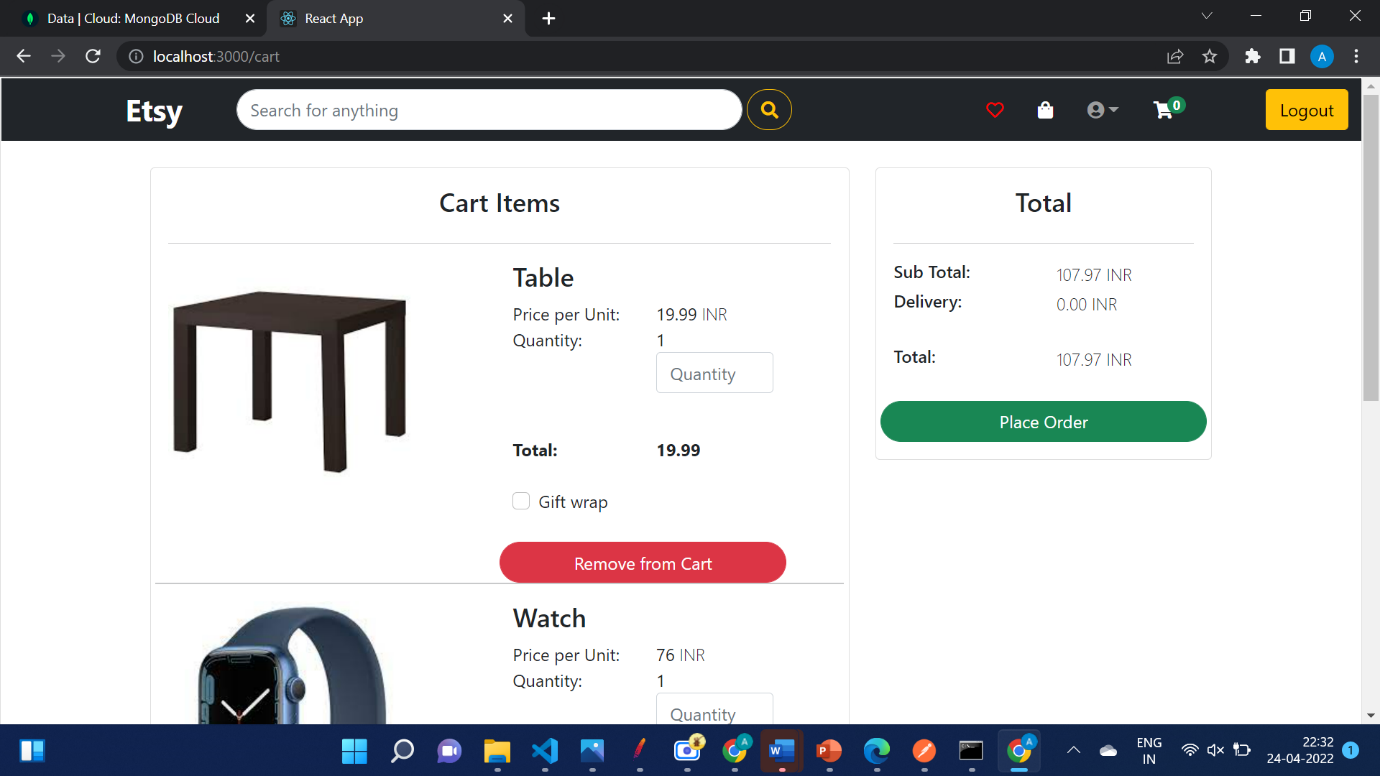
****

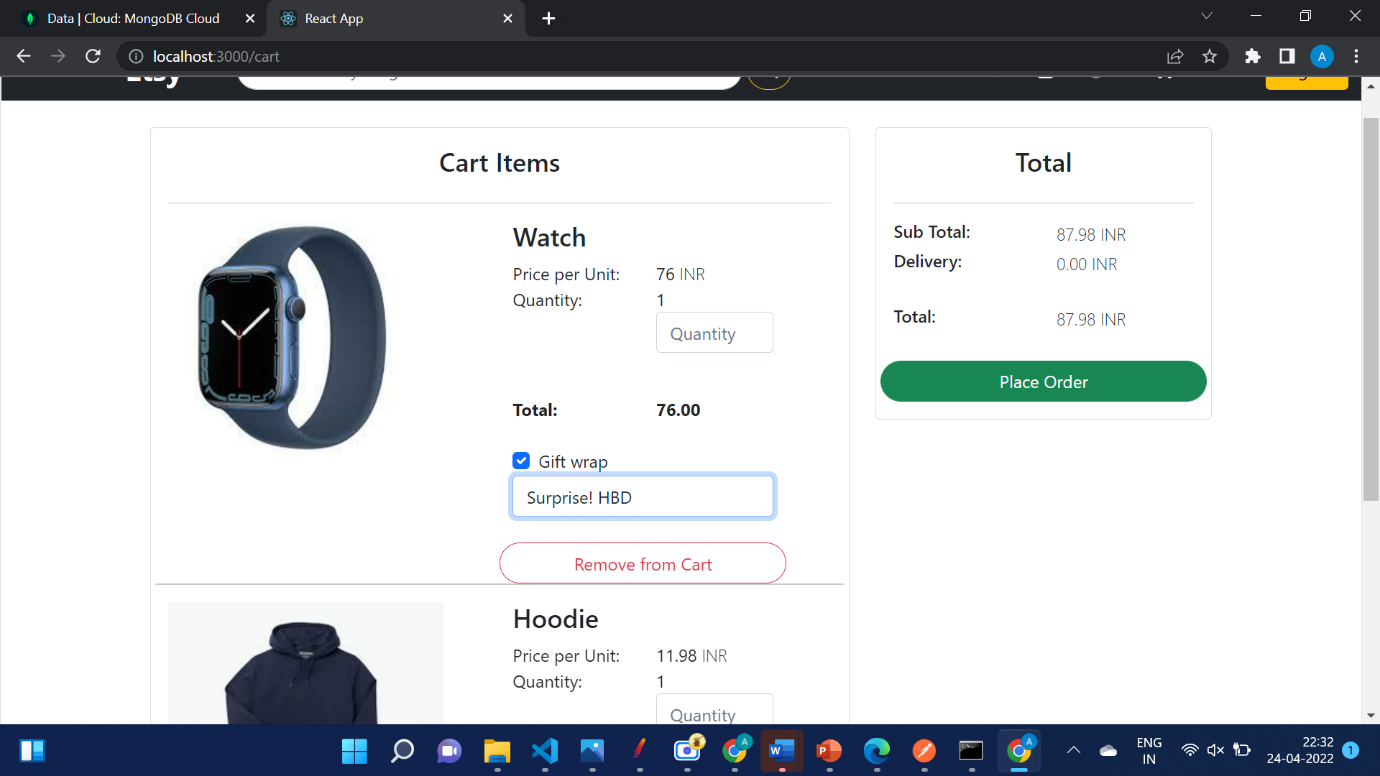
****

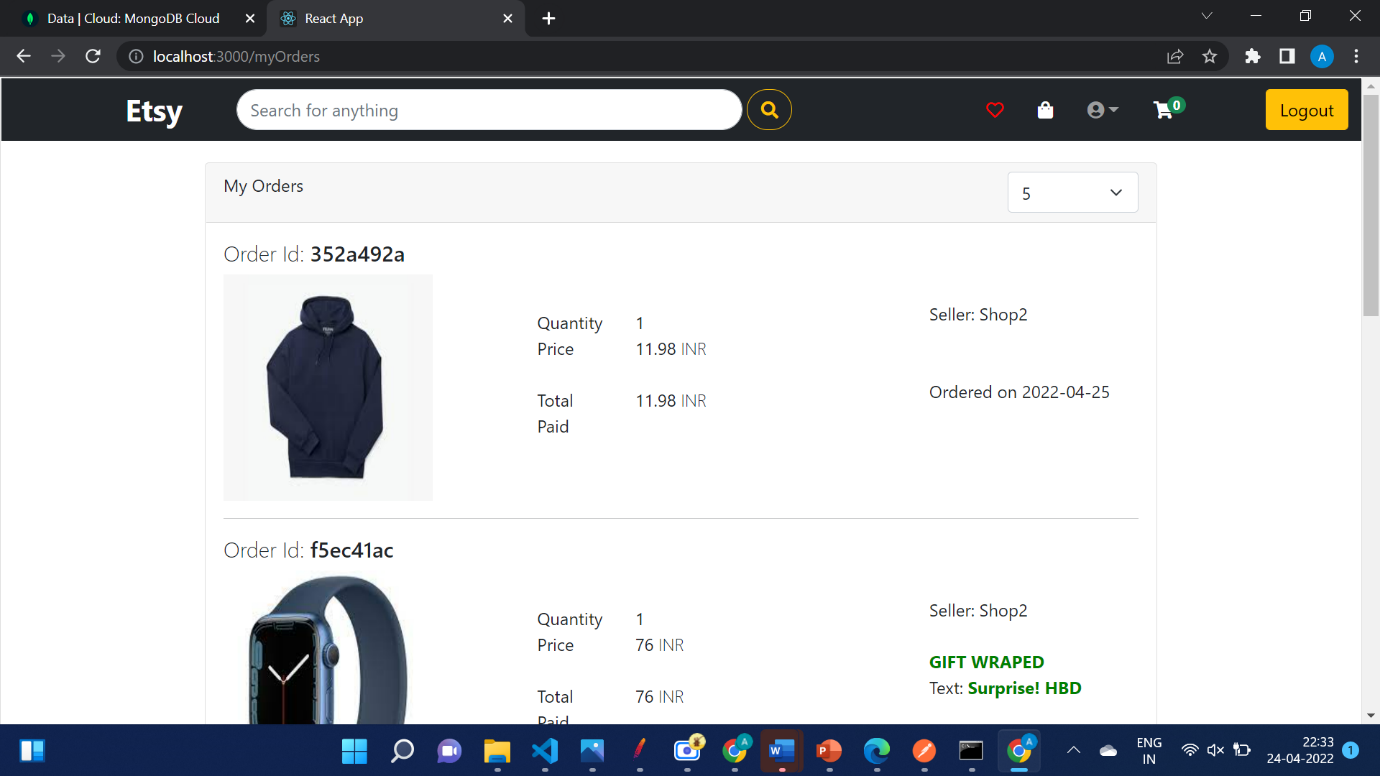
****

****

****

****

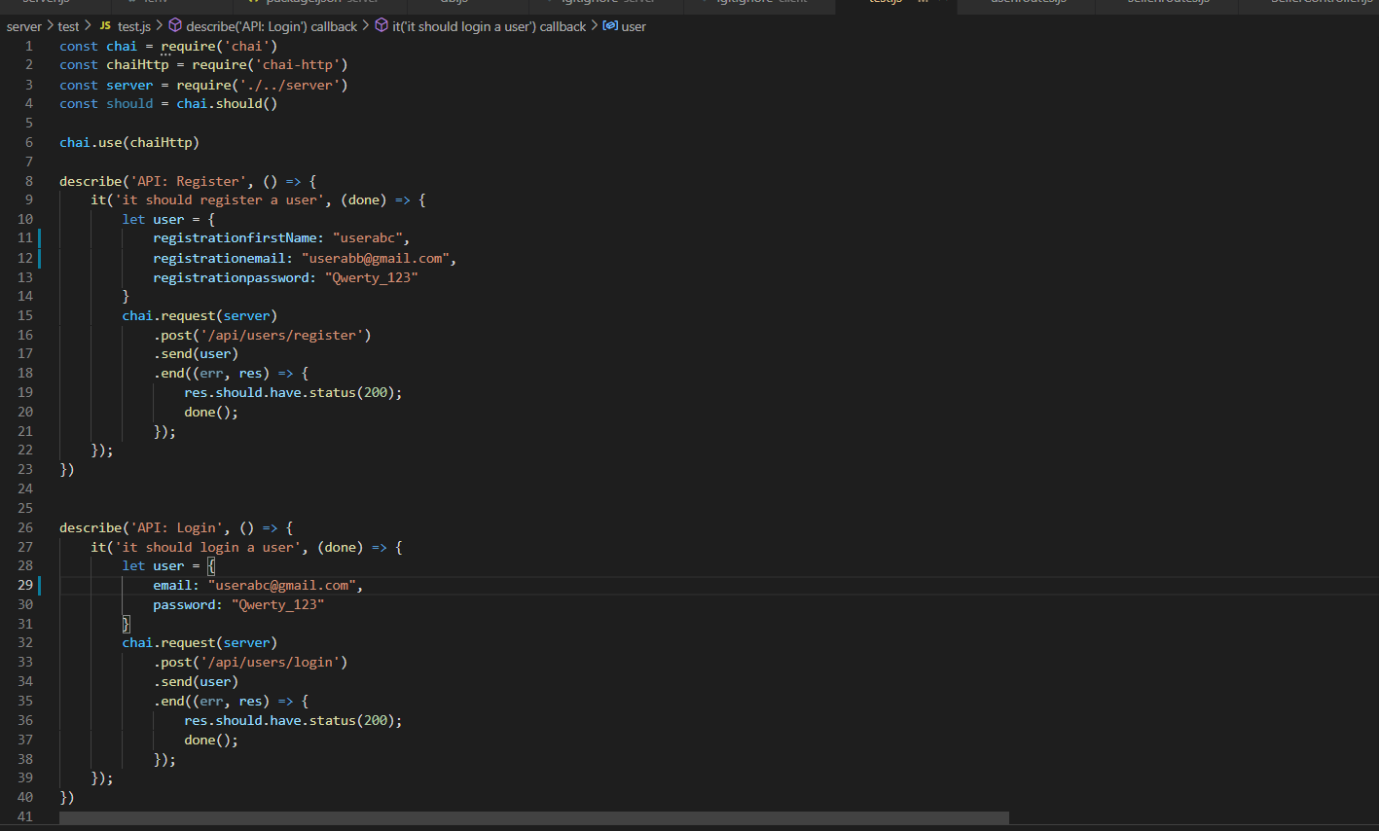
****

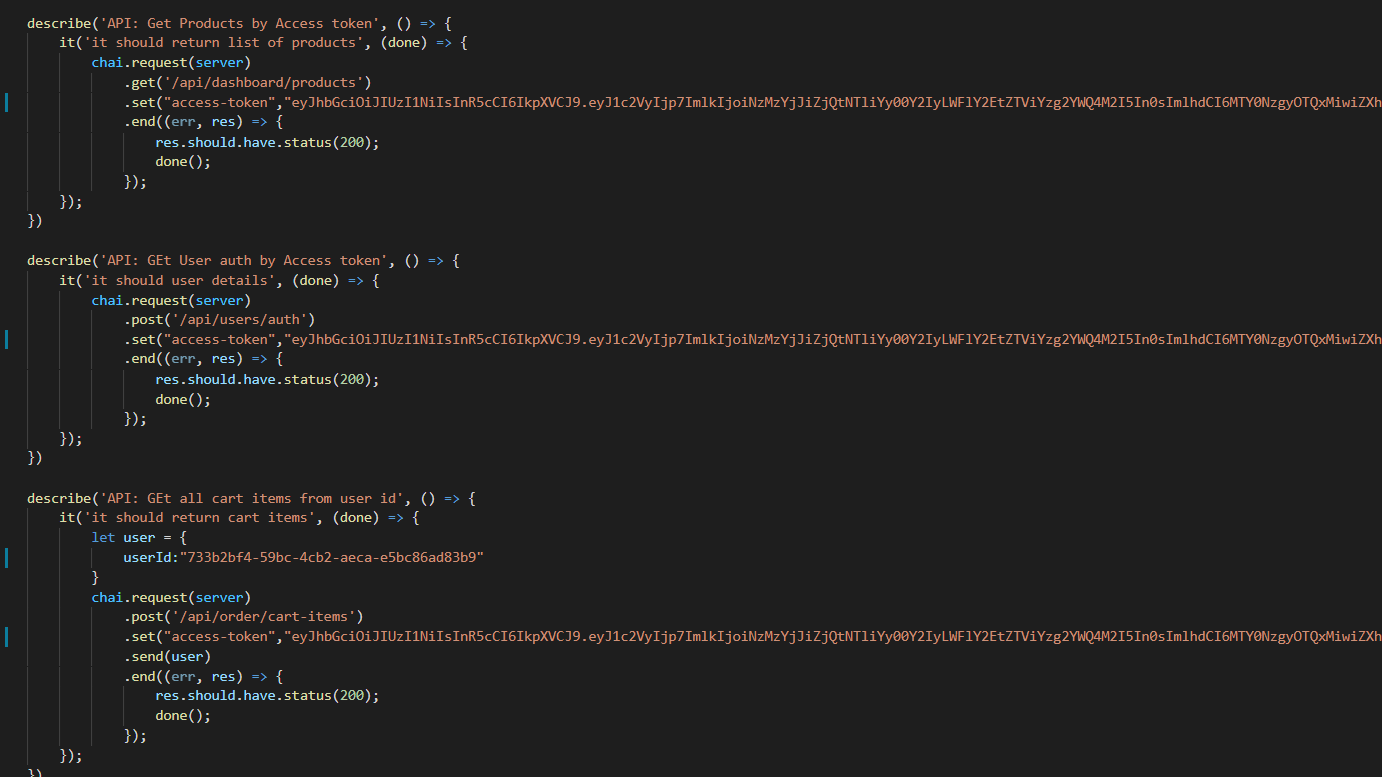
****

**Performance**

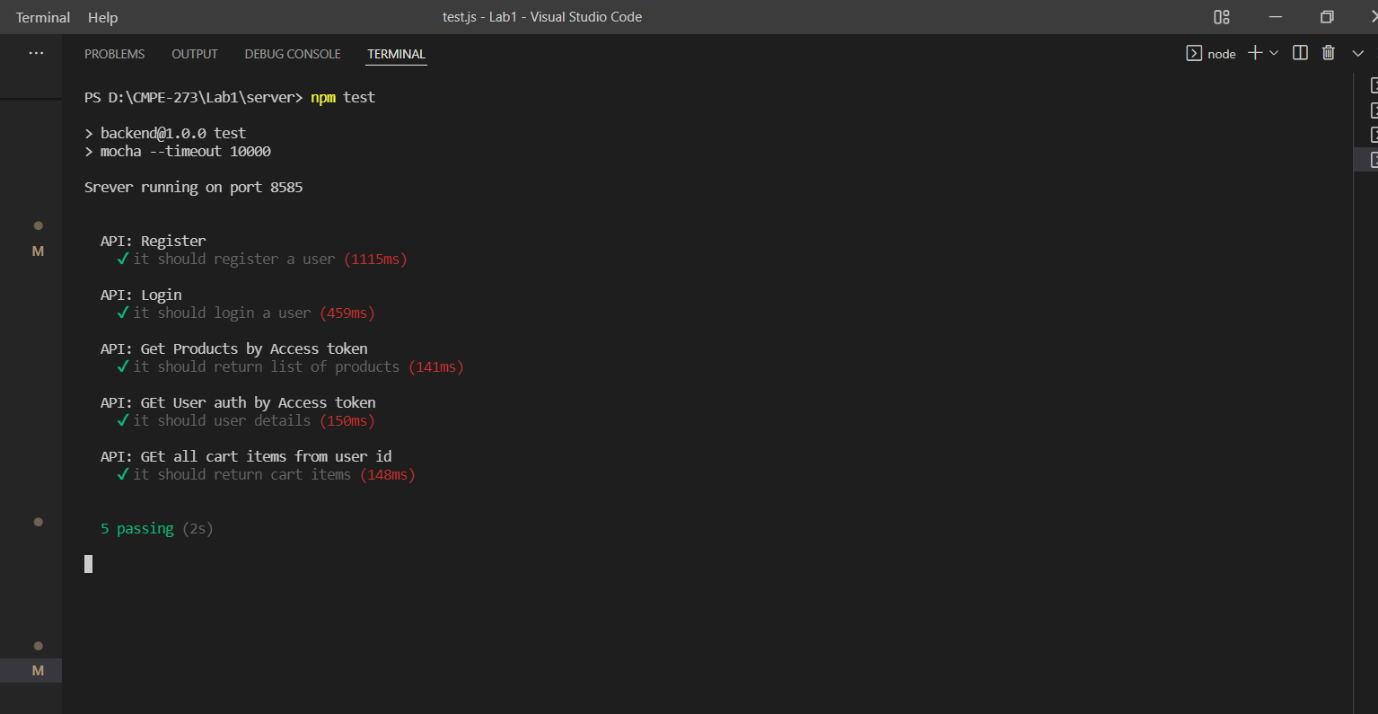
**Mocha Testing**

*Code***:**

****

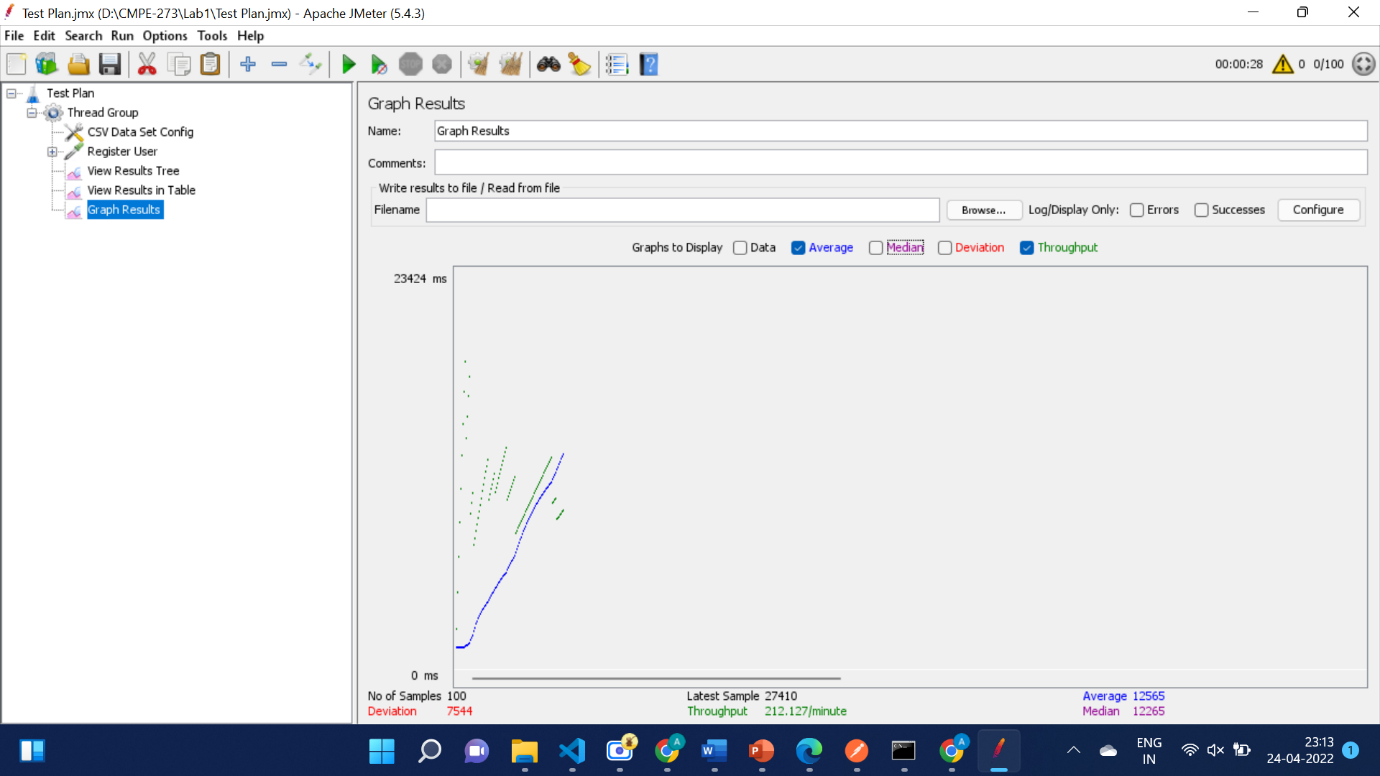
****

*Mocha Test Results:*

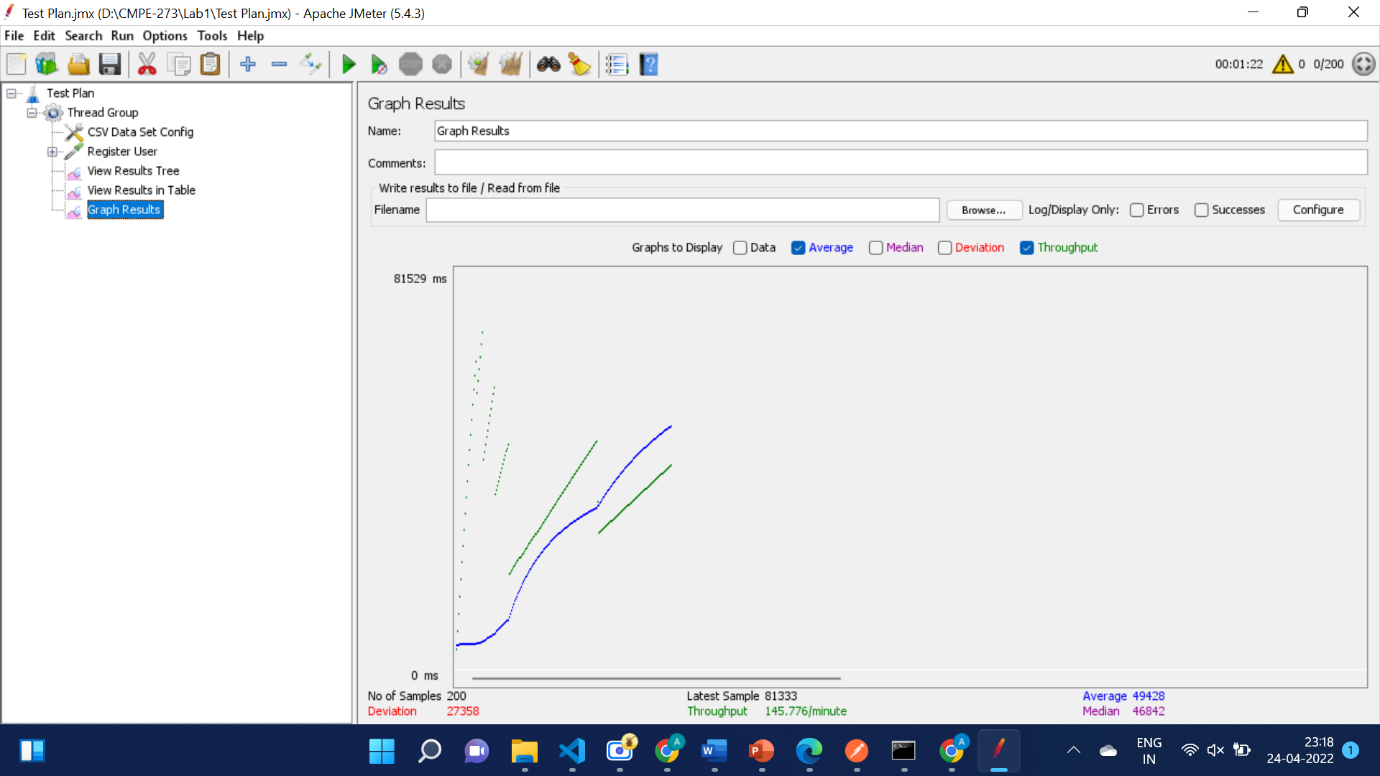
****

**Jmeter Testing:**

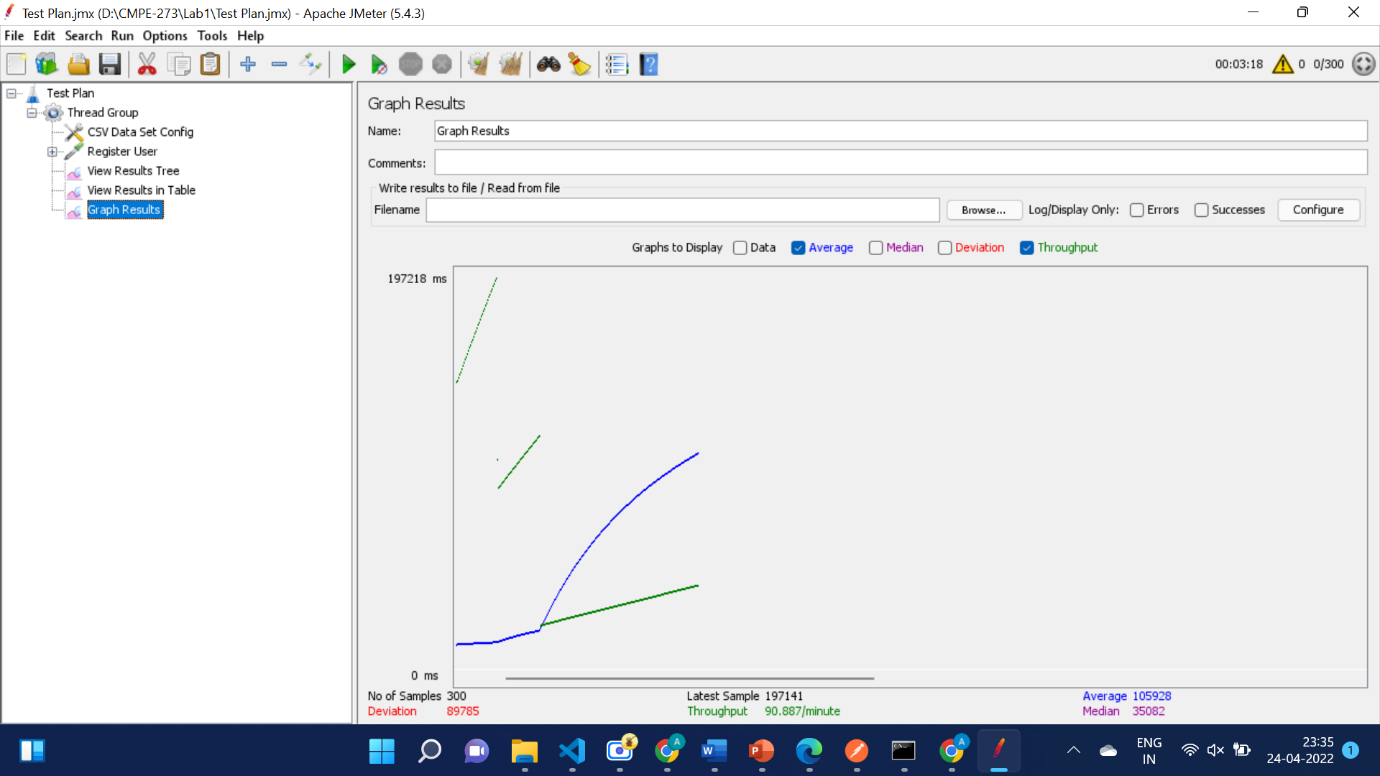
1. With Kafka:
2. 100 Users



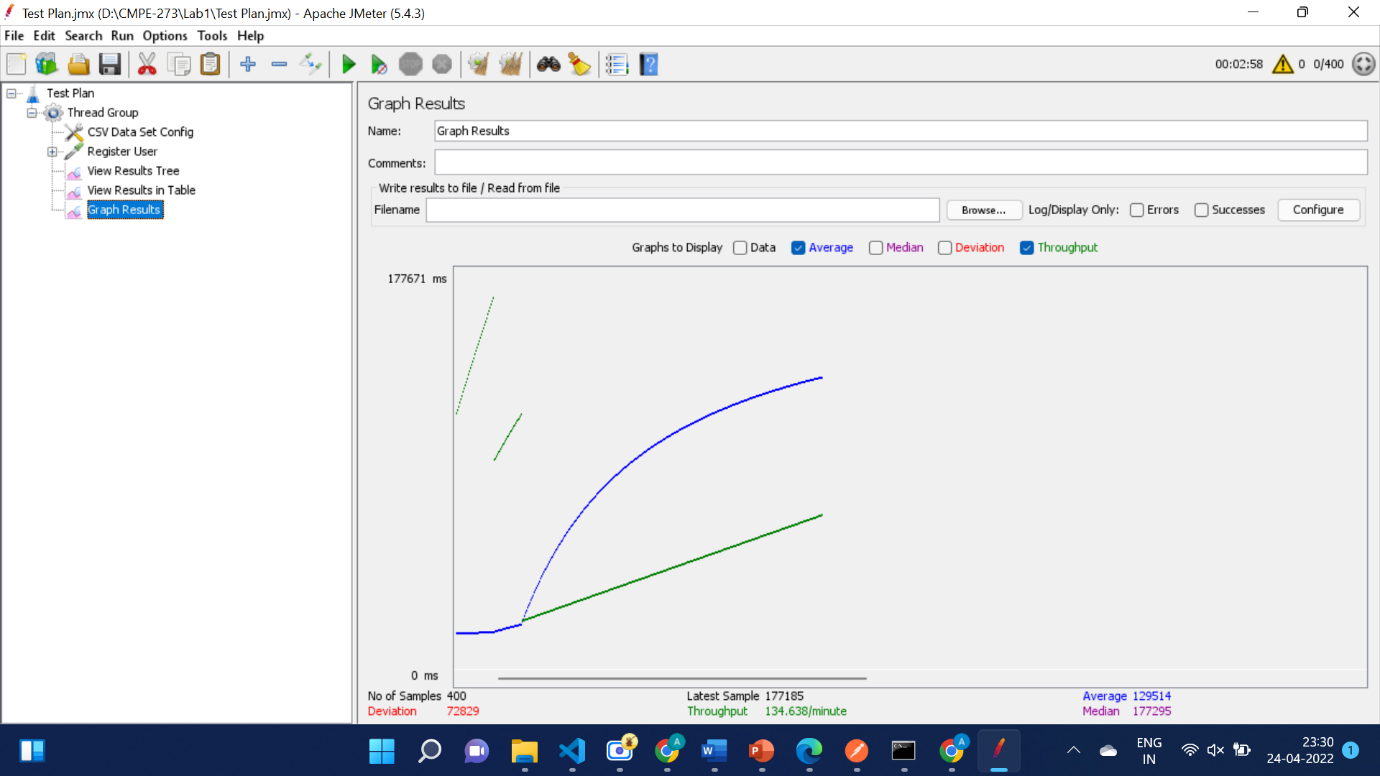
1. 200 Users



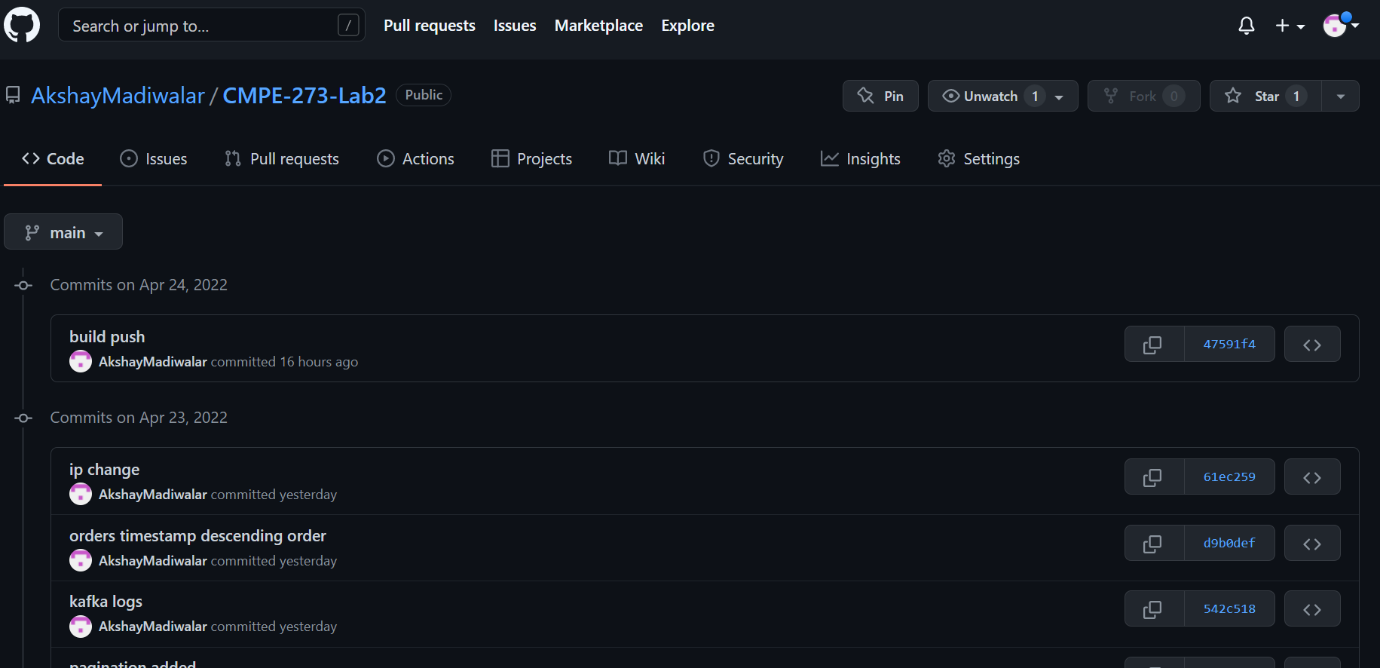
1. 300 Users



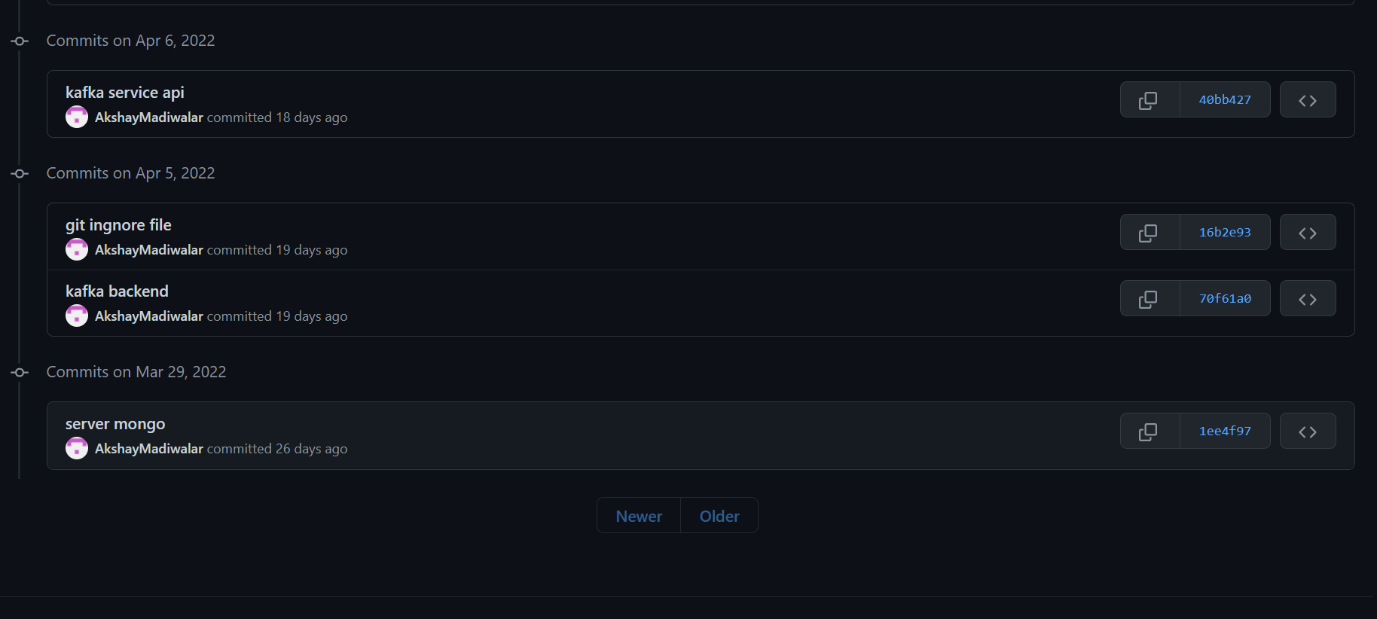
1. 400 Users



**Git History for ETSY**

****

****

****

**Questions**

1. **Compare the passport authentication process with the authentication process used in Lab1.**

Cookie based Authentication:

Cookies are used to handle the user session information in this authentication mechanism. After a successful login, cookies are set on the server side to do this. The frontend then uses these cookies to authenticate the user and grant access to the protected data/pages.

Passport JS Authentication:

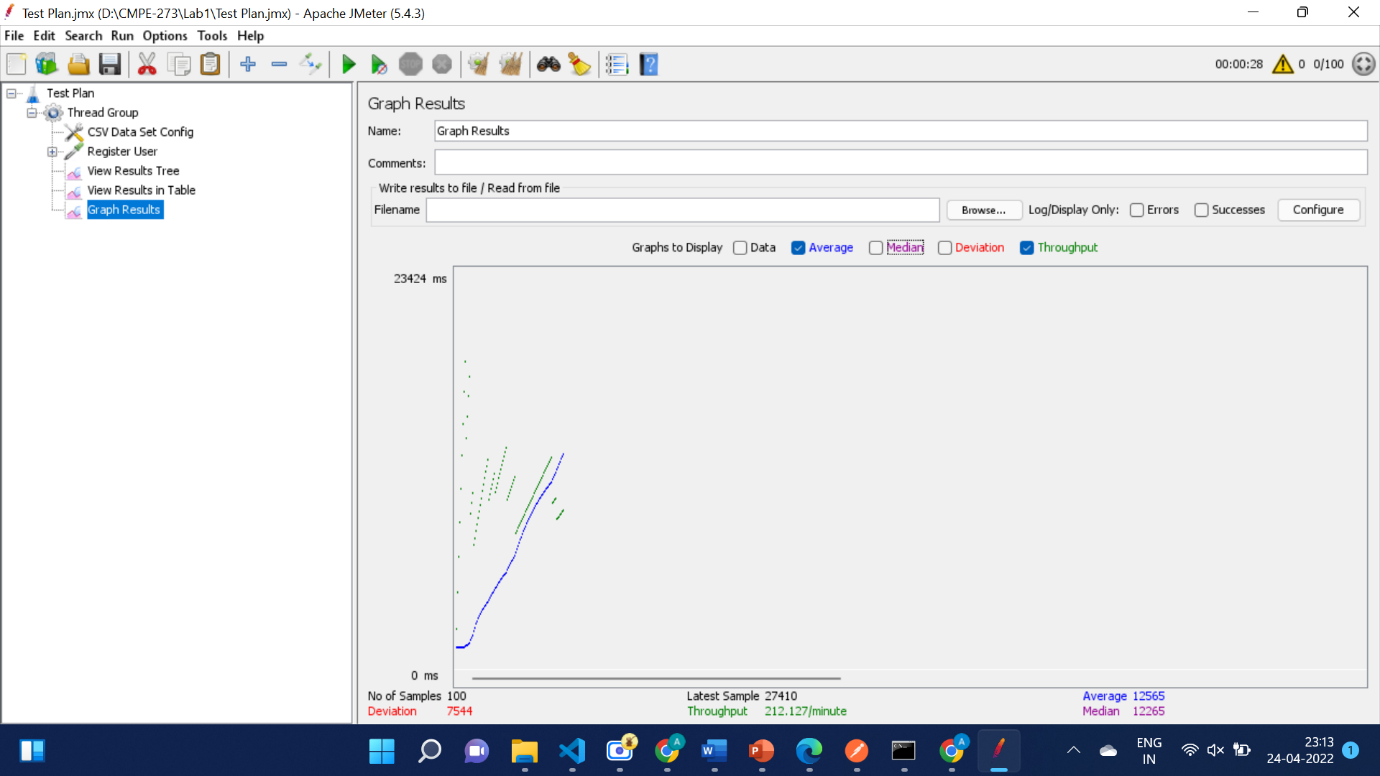
JWT authentication is a server-side authentication mechanism that generates a JSON Web Token (JWT) using the user's information and a secret key. This token is then saved in the client browser's local storage. It is then passed along with each client request. Each request is initially tested to determine if the token is a valid JWT token by decoding it with the same secret. When a user signs out or when the cookie's expiration date has passed, the cookies are deleted.

Passport.js is a Node.js authentication middleware that supports a vast range of authentication schemes. The passport-jwt technique aids in the JWT authentication process. Passport includes an authenticate method that aids in token authentication and is invoked for each request.

1. **Compare the performance with and without Kafka. Explain in detail the reason for the difference in performance.**

Kafka is a messaging queue system in which a producer can send messages and a consumer can subscribe to a topic and read the messages sent by the producer. The key advantage is that it separates the customer from the production, allowing both parties to work at their own pace.

The backend database system and the API server are separated in this system by Kafka. As a result, the API server serves as a producer, sending data request messages to the various Kafka topics. The backend database server functions as a consumer, consuming numerous messages for various topics. It gets the data from MongoDB and sends the output or error to the response topic, which the API server consumes.



Result with Kafka: Throughput: 212 requests / minute.

Advantages of Kafka:

1. High Availability: Because the consumer and producer are decoupled, if one instance of the consumer fails, another will pick up and consume the message. Furthermore, Kafka can be distributed across numerous clusters, resulting in high queue availability.
2. Durability: Kafka assures that a message will be sent at least once, reducing data loss in the event of a high load and providing the system with great durability.
3. Scalability: We can scale consumer and producer services horizontally to meet demand and improve performance.
4. **If given an option to implement MySQL and MongoDb in your application, specify which part of the application you will store in MongoDB and MySQL respectively.**

MySQL is a Relational Database System (RDBMS) that is used to store data that meets the following criteria:

Data is structured and less likely to change in the future.

• ACID characteristics are strictly enforced in data.

MongoDB is better suited for the following data types:

• Unstructured and large-volume data necessitates great scalability or caching.

In MySQL, I would store the following data for our Etsy application:

• Products and Orders data (Structured data) in MySQL database.

In Mongo DB, I would like to store:

• Users and favorites relationship document.