Etsy Prototype - Lab 2

Snigdha Chaturvedi | 015957435

Youtube Demo: https://www.youtube.com/watch?v=-Wn5MnilLVU
Git Repository: https://github.com/Snichat97/EtsyPrivateAWS

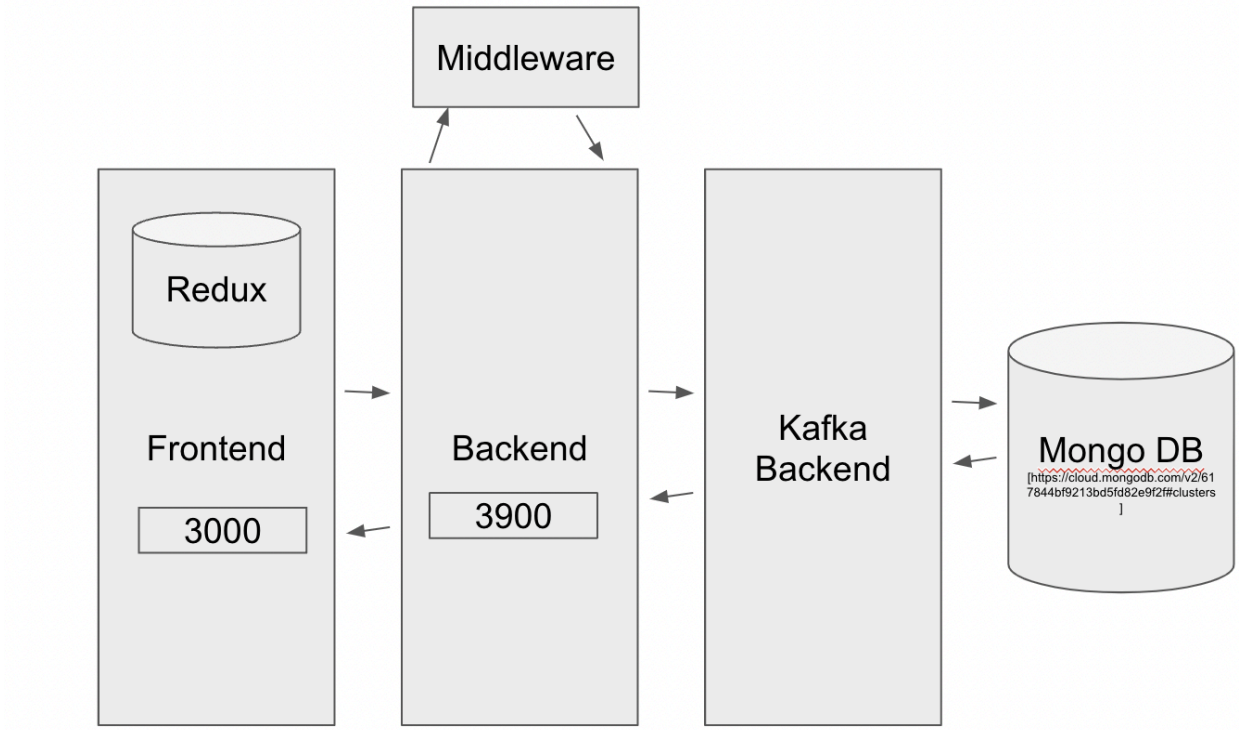
Goals and Purpose:

My goals from this project are to design and deploy an end-to-end application and explore in-depth backend API construction, various deployment strategies, and using the Redux library for state management across multiple components. Further, in Lab 2, I have explored the Kafka messaging queue between the backend and the database which enables handling a high number of requests. Also migrating the database from MySQL to MongoDB helped me to identify the differences between NoSQL and Relational Databases and understand in which use case which database can be leveraged. The system is a prototype of the “Etsy” Application which allows users to buy products and host their own shops.

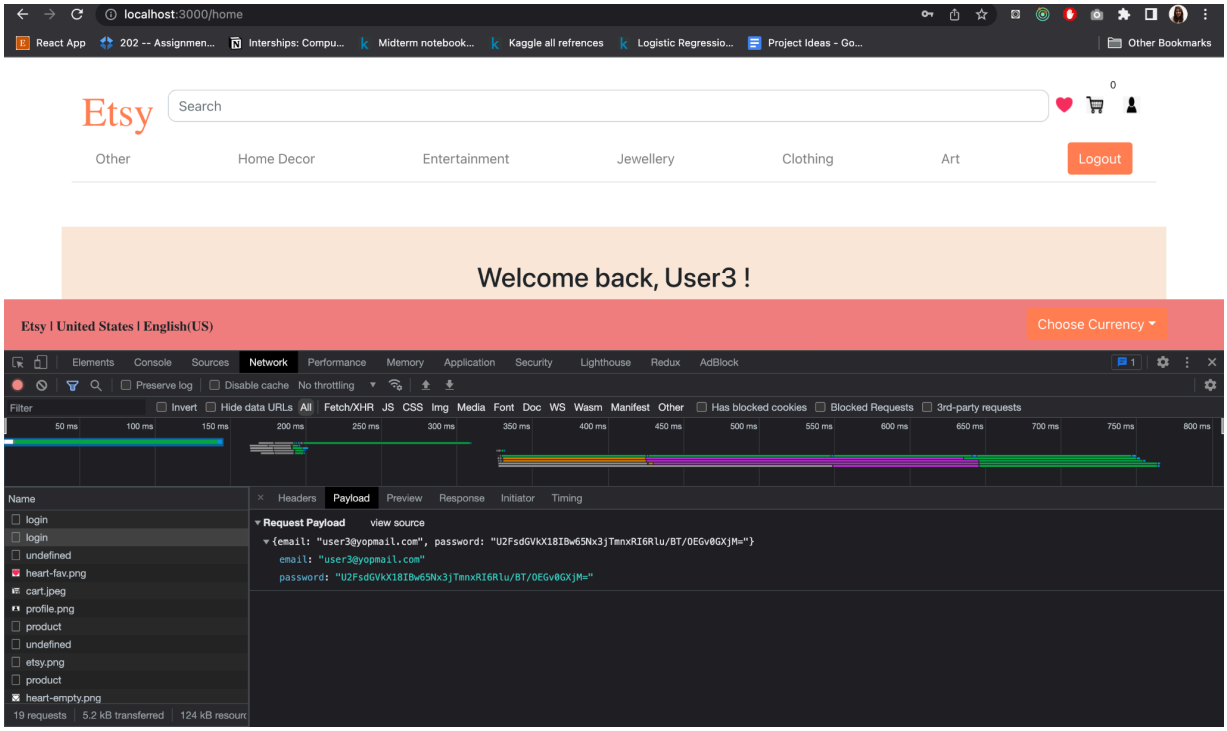
Salient features of the System:

The application uses ReactJS for the frontend and NodeJs running on Express Server for the backend where the call is sent to the middleware to test the authentication token. On verification, it is sent to the kafka-backend.

An AWS EC2 instance holds the complete application. On port 3000 the frontend of the application is hosted and on 3900 the backend is running. Using the mongoose library it connects to the MongoDB database. The MongoDB server is running on Atlas (https://cloud.mongodb.com/)

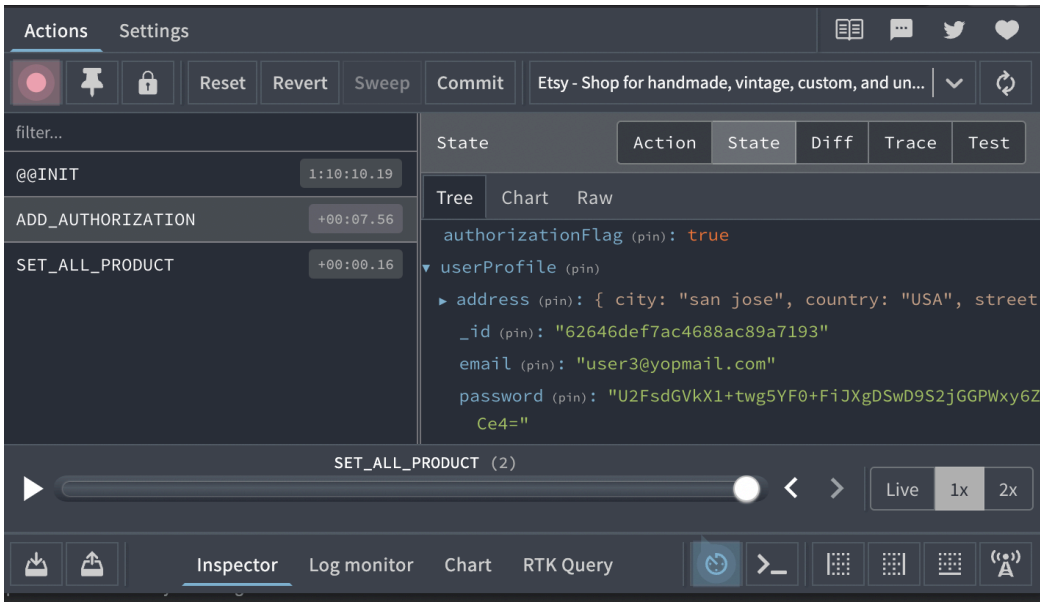


Screen Previews

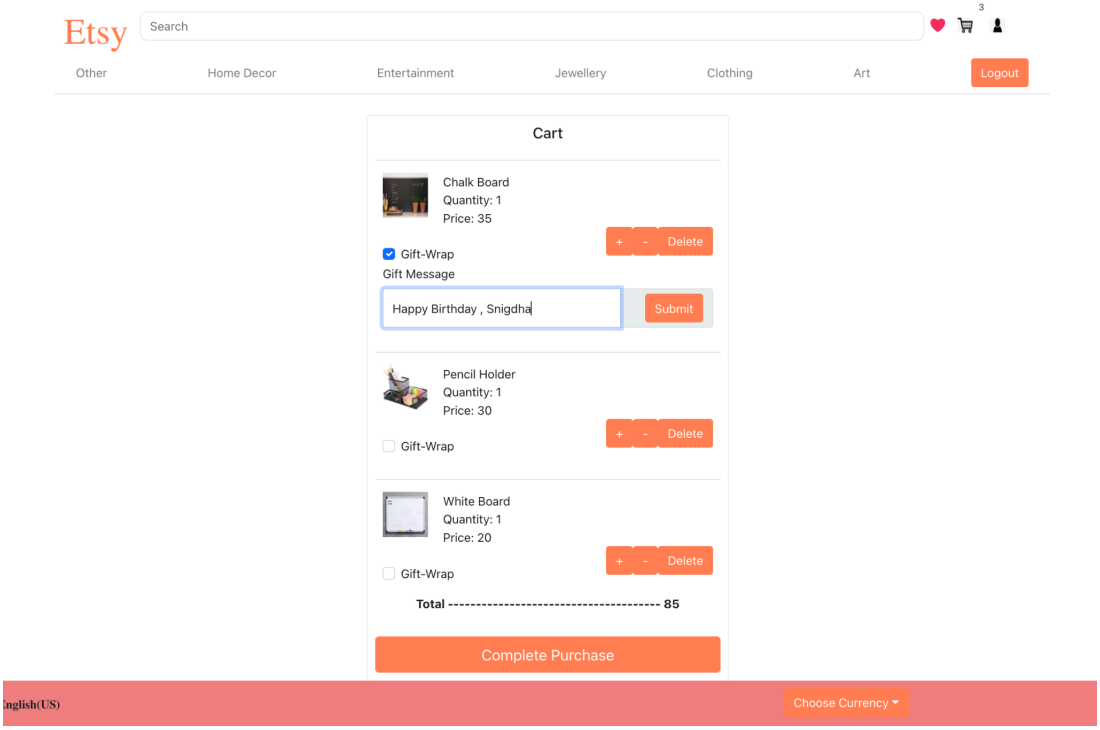


On the Login Call, the Authorization token is returned

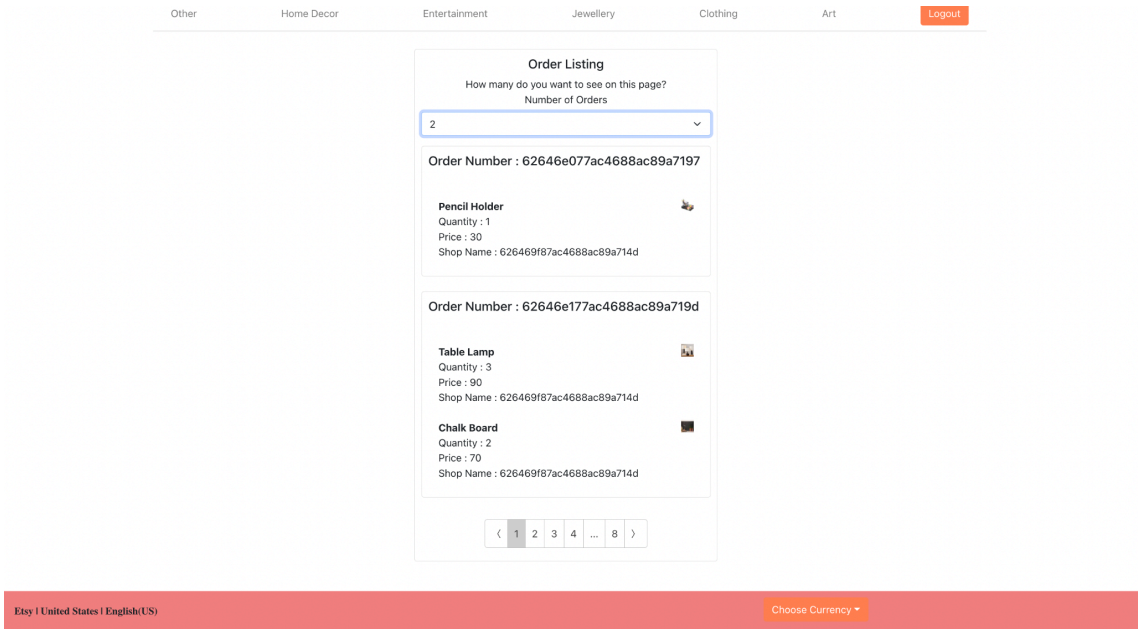
Storing the User Profile in Redux



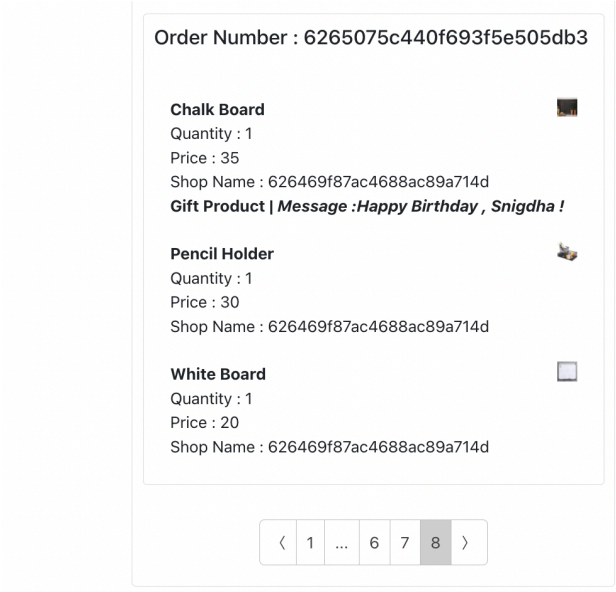
Gift Wrapping and Gift messaging option



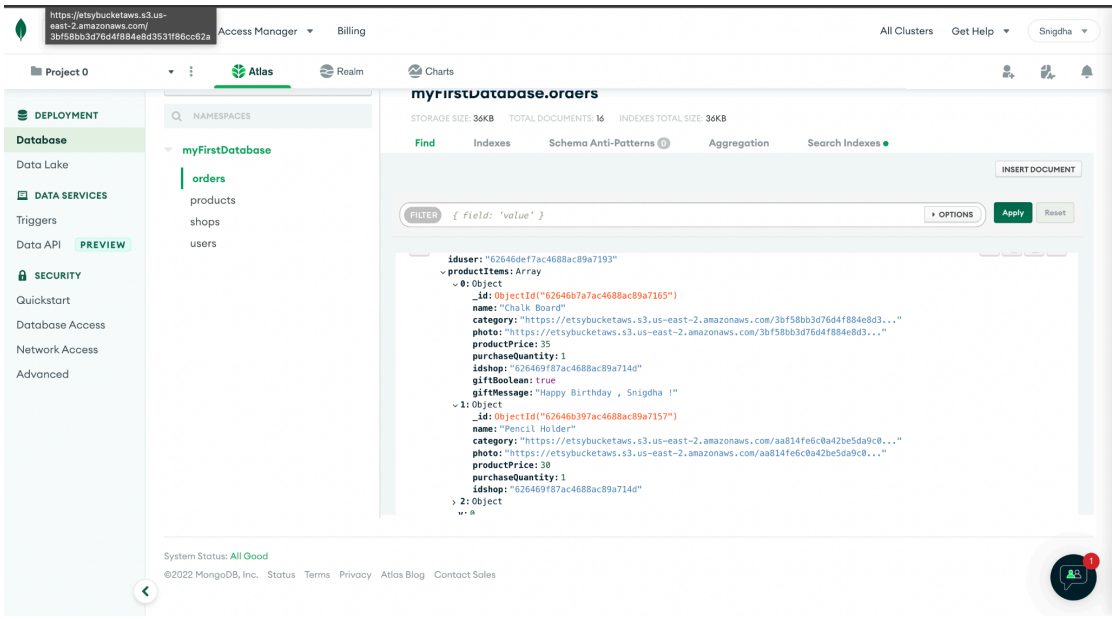
Order Pagination



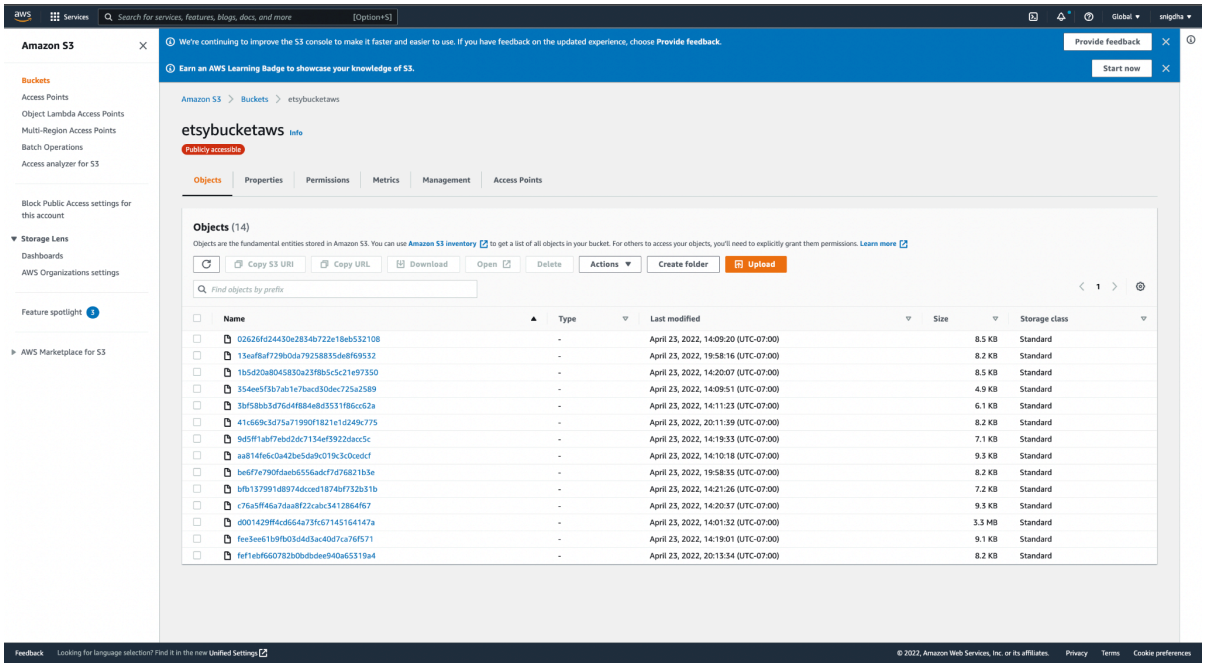
Order History - With Gift Wrap, Customized message, and Order Pagination



MongoDB - Deployed on Atlas

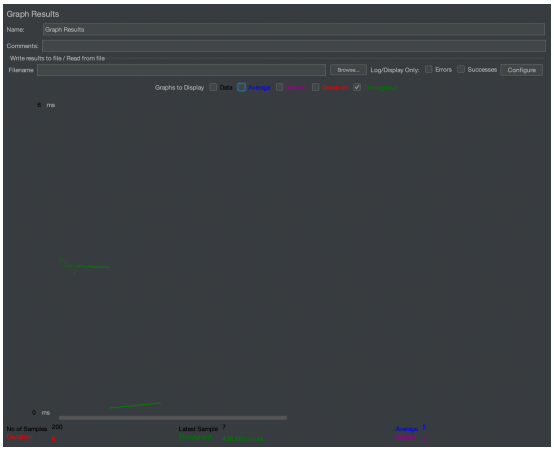
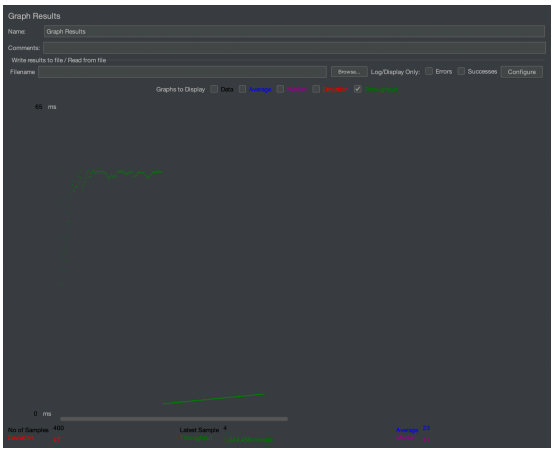
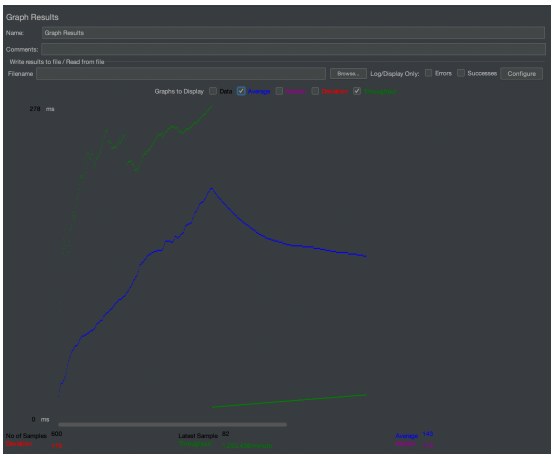


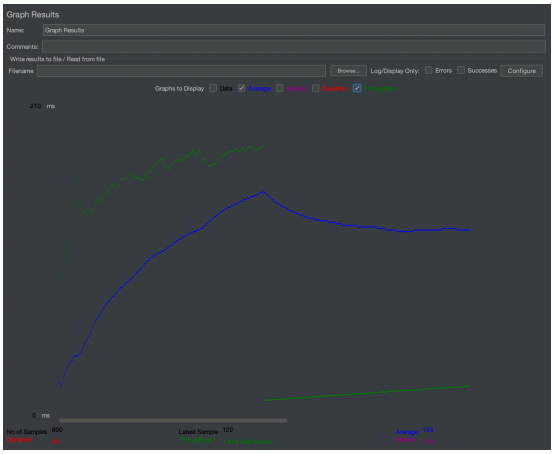
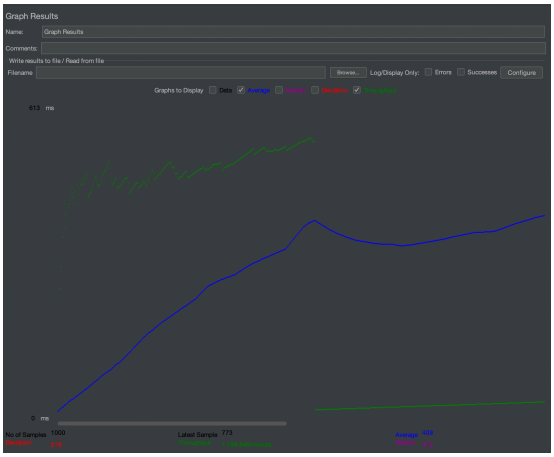
Images deployed on S3



Testing

1. JMeter Testing

Number of congruent users	Graph pooling vs single connection
100 Users	
200 Users	
300 Users	

400 Users	
500 Users	

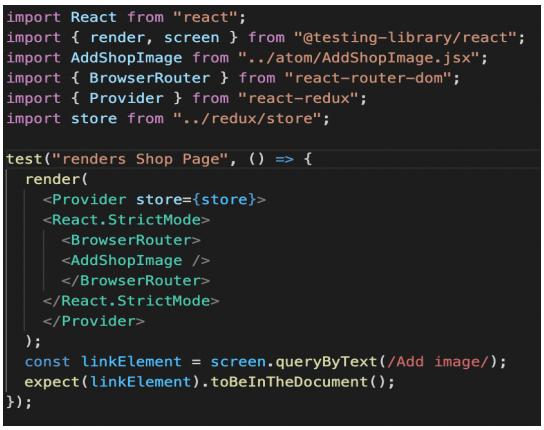
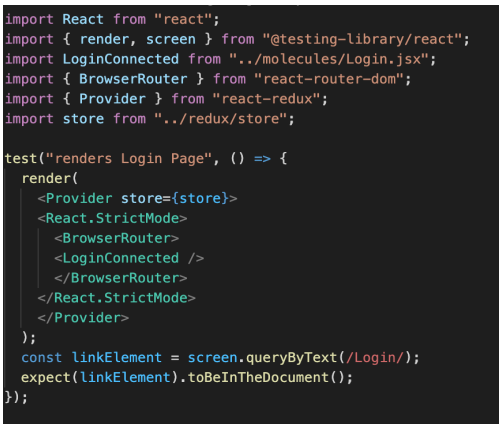
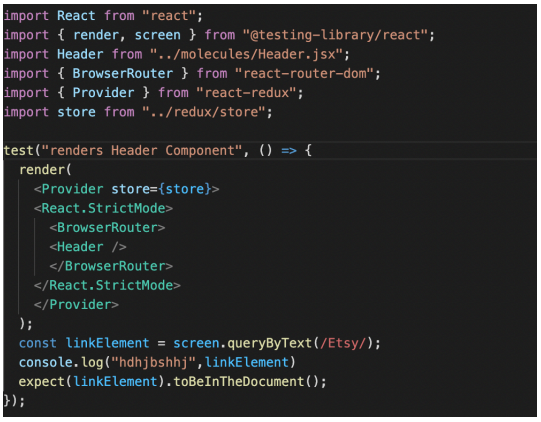
2. Mocha Testing


```
const axios = require("axios");

const assert = require("assert");

const apiUrl = "http://localhost:3900/api";

describe("POST - check for sign in", () => {
  it("/user/login", (done) => {
    axios
      .post(apiUrl + "/user/login", { email: "snigdha.chaturvedi@sjsu.edu", password: "snigdha" })
      .then((response) => {
        console.log(response.data);
        assert.equal(response.status, 200);
        done();
      })
      .catch((err) => {
        done(err);
      });
  });
});
```

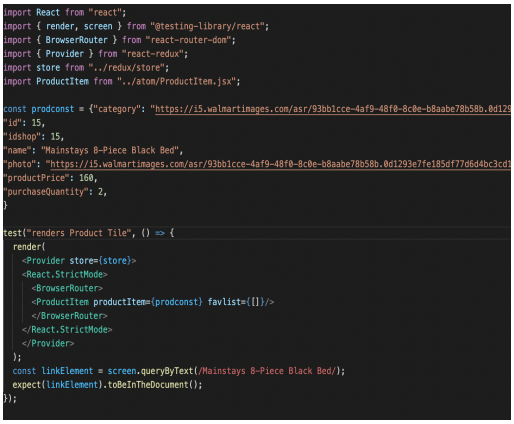
```
describe("GET - get customer details by id", () => {
  it("/user/25", (done) => {
    axios
      .get(apiUrl + "/user/25")
      .then((response) => {
        console.log(response.data);
        assert.equal(response.status, 200);
        done();
      })
      .catch((err) => {
        done(err);
      });
  });
});

describe("GET - product details", () => {
  it("/product/14", (done) => {
    axios
      .get(apiUrl + "/product/14")
      .then((response) => {
        console.log(response.data);
        assert.equal(response.status, 200);
        done();
      })
      .catch((err) => {
        done(err);
      });
  });
});

describe("GET - shop details", () => {
  it("/shop/16", (done) => {
    axios
      .get(apiUrl + "/shop/16")
      .then((response) => {
        console.log(response.data);
        assert.equal(response.status, 200);
        done();
      })
      .catch((err) => {
        done(err);
      });
  });
});
```

```
describe("GET - user's shop ", () => {
  it("/user/25", (done) => {
    axios
      .get(apiUrl + "/user/25")
      .then((response) => {
        console.log(response.data);
        assert.equal(response.status, 200);
        done();
      })
      .catch((err) => {
        done(err);
      });
  });
});
```

3. React Testing

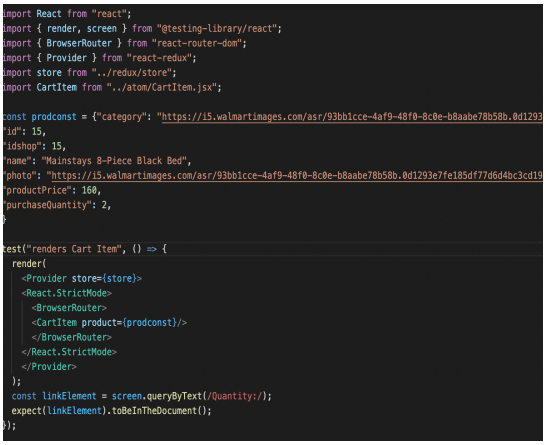
Component/Page	React Testing Library
Shop Page	
Login Page	
Header	

Product Tile



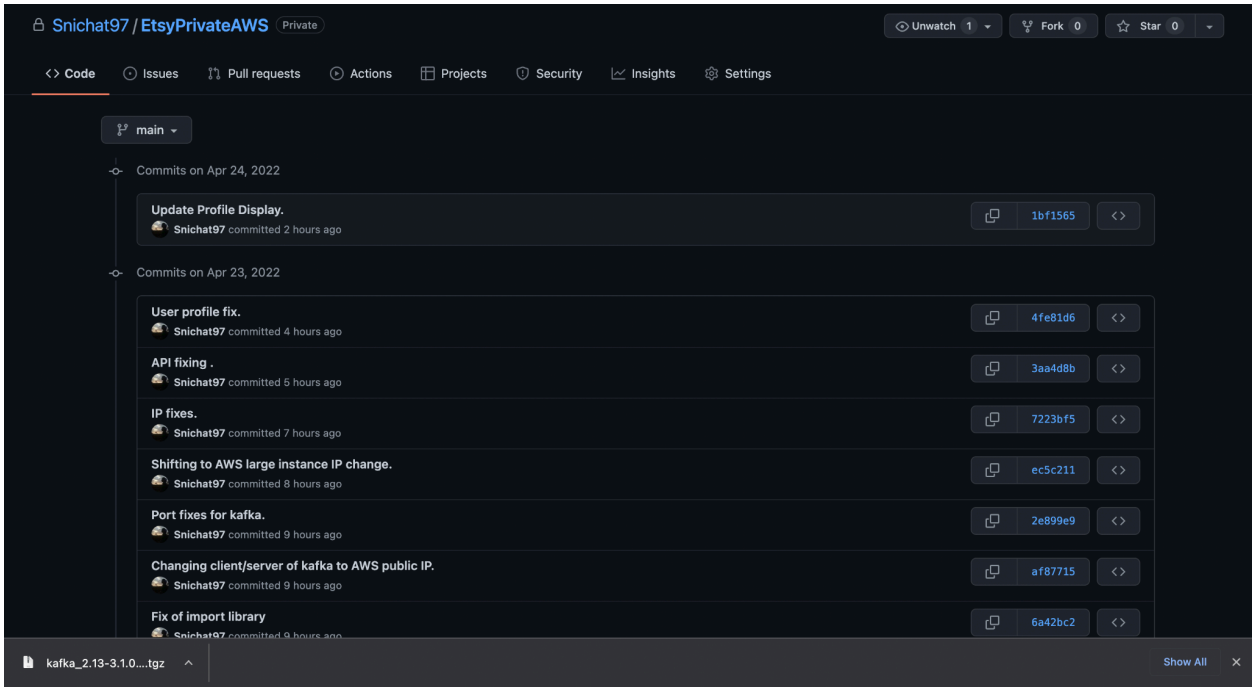
```
import React from "react";
import { render, screen } from "@testing-library/react";
import { BrowserRouter } from "react-router-dom";
import { Provider } from "react-redux";
import store from "../redux/store";
import ProductItem from "../atom/ProductItem.jsx";

const prodconst = {"category": "https://i5.walmartimages.com/asr/93bb1cce-4af9-48f0-8c0e-b8aabe78b58b.0d129
"id": 15,
"idshop": 15,
"name": "Mainstays 8-Piece Black Bed",
"photo": "https://i5.walmartimages.com/asr/93bb1cce-4af9-48f0-8c0e-b8aabe78b58b.0d1293e7fe185df77d6d4bc3cd1
"productPrice": 160,
"purchaseQuantity": 2,
}

test("renders Product Tile", () => {
  render(
    <Provider store={store}>
    <React.StrictMode>
      <BrowserRouter>
      <ProductItem productItem={prodconst} favlist={[]}/>
      </BrowserRouter>
    </React.StrictMode>
    </Provider>
  );
  const linkElement = screen.queryByText(/Mainstays 8-Piece Black Bed/);
  expect(linkElement).toBeInTheDocument();
});
```

Cart Item



```
import React from "react";
import { render, screen } from "@testing-library/react";
import { BrowserRouter } from "react-router-dom";
import { Provider } from "react-redux";
import store from "../redux/store";
import CartItem from "../atom/CartItem.jsx";

const prodconst = {"category": "https://i5.walmartimages.com/asr/93bb1cce-4af9-48f0-8c0e-b8aabe78b58b.0d1293
"id": 15,
"idshop": 15,
"name": "Mainstays 8-Piece Black Bed",
"photo": "https://i5.walmartimages.com/asr/93bb1cce-4af9-48f0-8c0e-b8aabe78b58b.0d1293e7fe185df77d6d4bc3cd19
"productPrice": 160,
"purchaseQuantity": 2,
}

test("renders Cart Item", () => {
  render(
    <Provider store={store}>
    <React.StrictMode>
      <BrowserRouter>
      <CartItem product={prodconst}/>
      </BrowserRouter>
    </React.StrictMode>
    </Provider>
  );
  const linkElement = screen.queryByText(/Quantity:/);
  expect(linkElement).toBeInTheDocument();
});
```

Git Commit history



Performance:

Performance testing was done using JMeter. Visible across all loads (100,200,300,400,500 users) the connection pooling of 500 connections outperformed the single connection.It is significantly notable in heavier loads.

With 100 user test case

- Connection Pooling | 0.3 ms average throughput
- Single connection | 3 ms average throughput

With 500 user test case

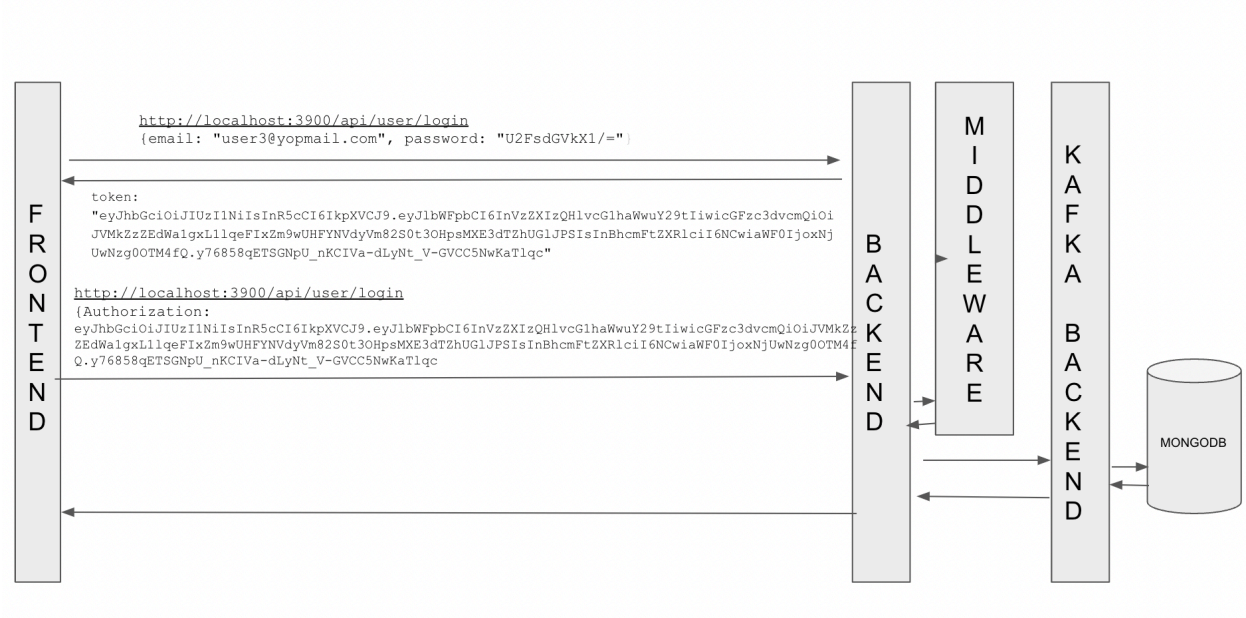
- Connection Pooling | 1 ms average throughput
- Single connection | 613 ms average throughput

Questions

Please find the questions for Lab2 which you have to answer in your lab report

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

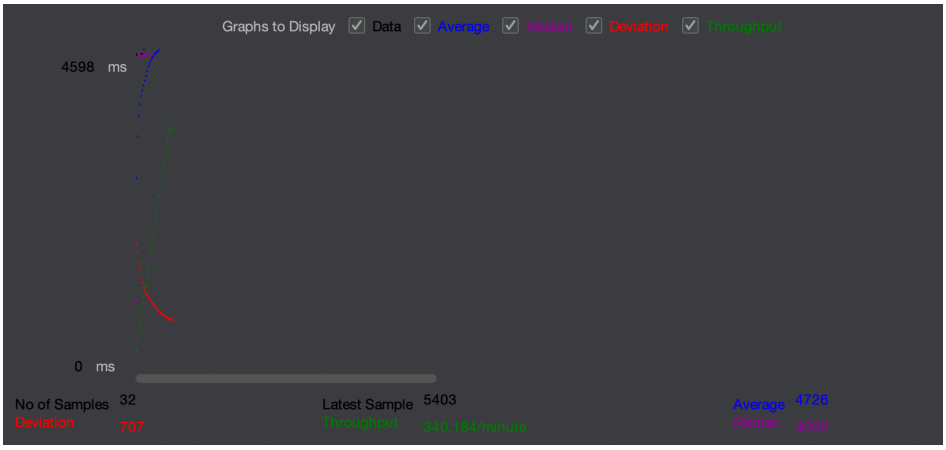
   In the current flow of the application when the user logs in or signs up for the first time an authentication token is returned, which is carried forward in all the subsequent API calls. Before the API is executed the call is sent to the middleware. If the token is valid the API is executed else an error is sent to the frontend.



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

   Kafka significantly enhances the performance of the Application.This is visible in various parameters :

	Kafka	Backend Directly with MongoDB
Throughput	4400.763/ min	340.184/min
Messages Dropped	0	Multiple queries resulted in unavailable server
Average Response Time	2948 ms	4726 ms



3. **Given an option to implement MySQL and MongoDB both in your application, specify which part of data of the application you will store in MongoDB and MySQL respectively**

   I would use MySQL for user, shop, and product tables as they all are following strict relational schema and for various functionalities. Also for these tables (favorite products: user - product, shop products: shop - product), join functionality is leveraged.

   For Orders, I would use MongoDB as it has a flexible schema that may or may not have Gift Boolean or Gift Message. It also has some old data maintained for products due to the dynamic schema MongoDB is more ideal for this usecase.