Etsy Prototype - Lab 3

Snigdha Chaturvedi | 015957435

Introduction: Goals and Purpose of the system

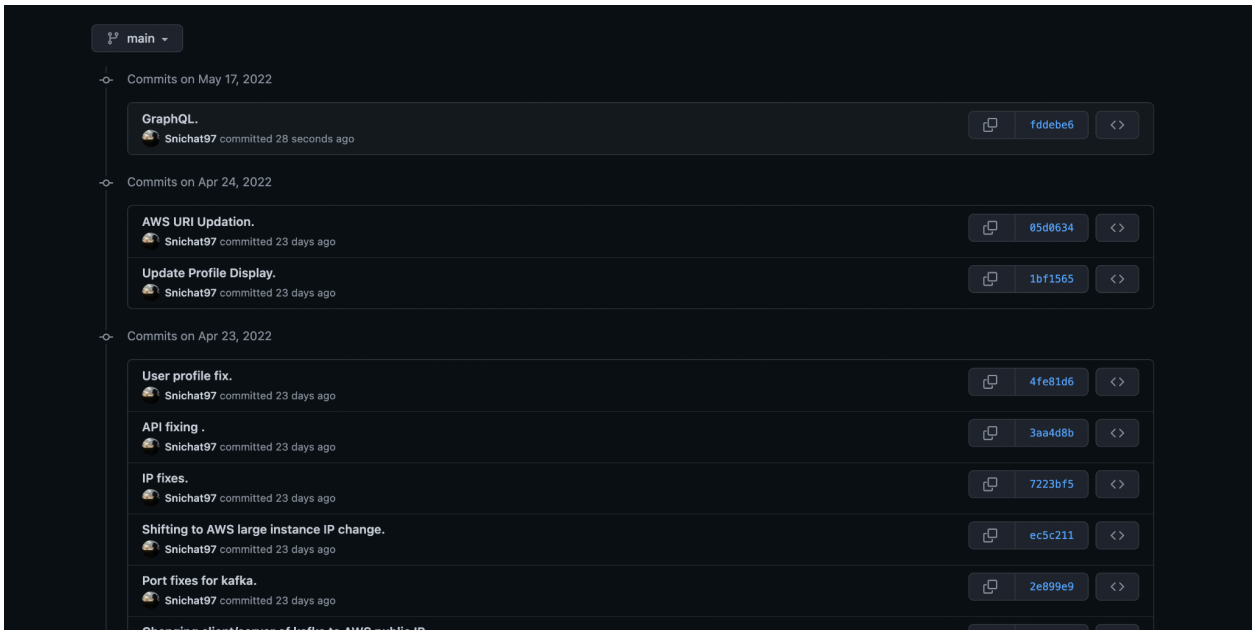
My goals from this project are to design and deploy an end-to-end application and explore in-depth backend API construction, various deployment strategies, and using the Redux library for state management across multiple components.

Further, in Lab 3, I have explored GraphQL and learned to design a backend that is lightweight and more easily scalable and can be used to derive only required and relevant data.

System Design for Lab3

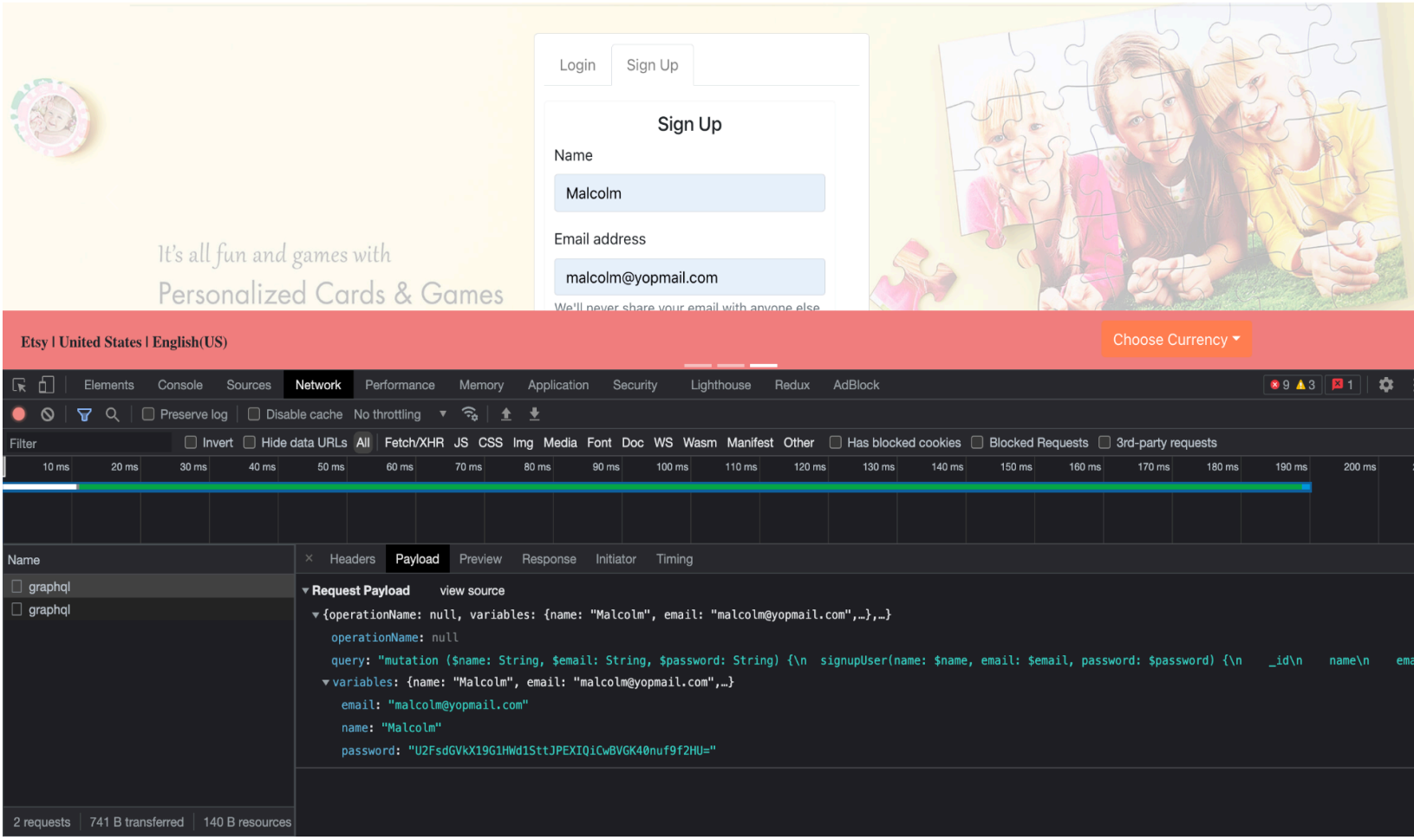
The application uses ReactJS and Apollo client for the frontend and NodeJs running on Express-GraphQL at the Backend. The backend uses mongoose library to connect to the MongoDB database. The MongoDB server is running on Atlas (https://cloud.mongodb.com/). Images are hosted on S3 (As done in previous labs)

Git Commit History



Lab Steps

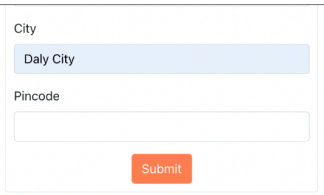
1. Create an account for user “Malcolm”.

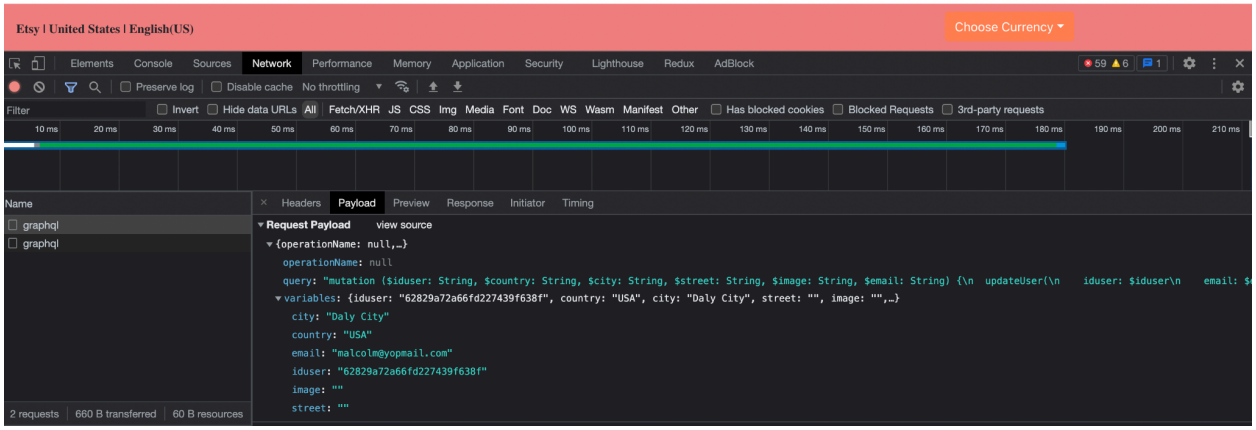


2. Paste the snapshot showing the mutation at the backend for Malcolm.

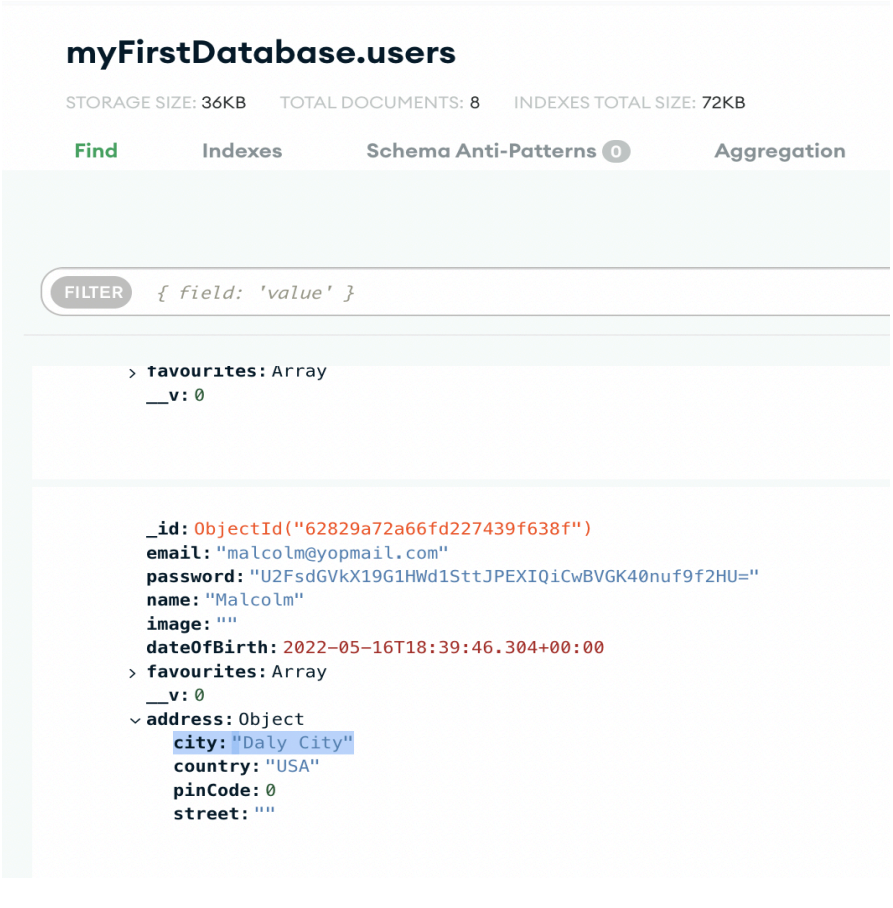
```
const Mutation = new GraphQLObjectType(
    {
    name: 'Mutation',
    fields: {
       signupUser: {
           type: UserType,
           args: {
               name: { type: GraphQLString },
               email:{ type: GraphQLString },
               password: { type: GraphQLString }
           },
           async resolve(parent, args) {
               let user = await User.findOne({ email: args.email });
               if (user){
                   return res.status(400).send("Username already exists");
               }

               user = new User({
               name: args.name,
               email: args.email,
               password: args.password,
               dateOfBirth:new Date().toISOString(),
               image : "",
               phoneNum : 0000000000,
               address:{}
               });

               await user.save();
               console.log(user)
               return user
           }
       },
```

3. Navigate to the user profile and update the city to “Daly City”.

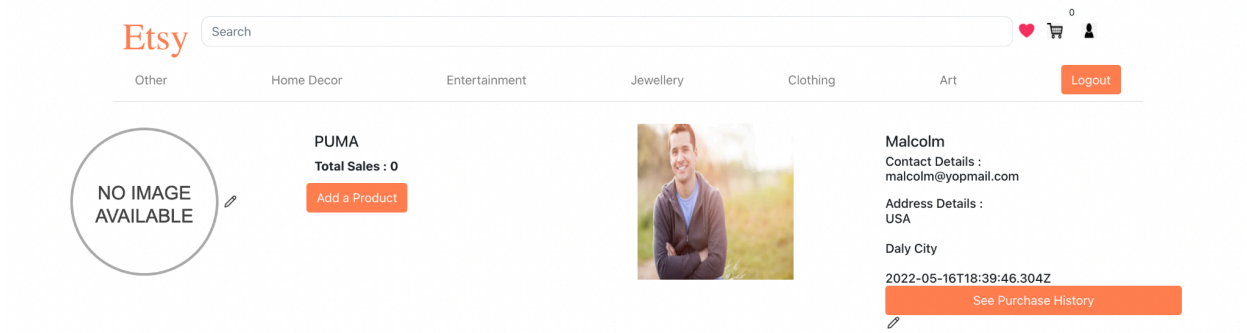


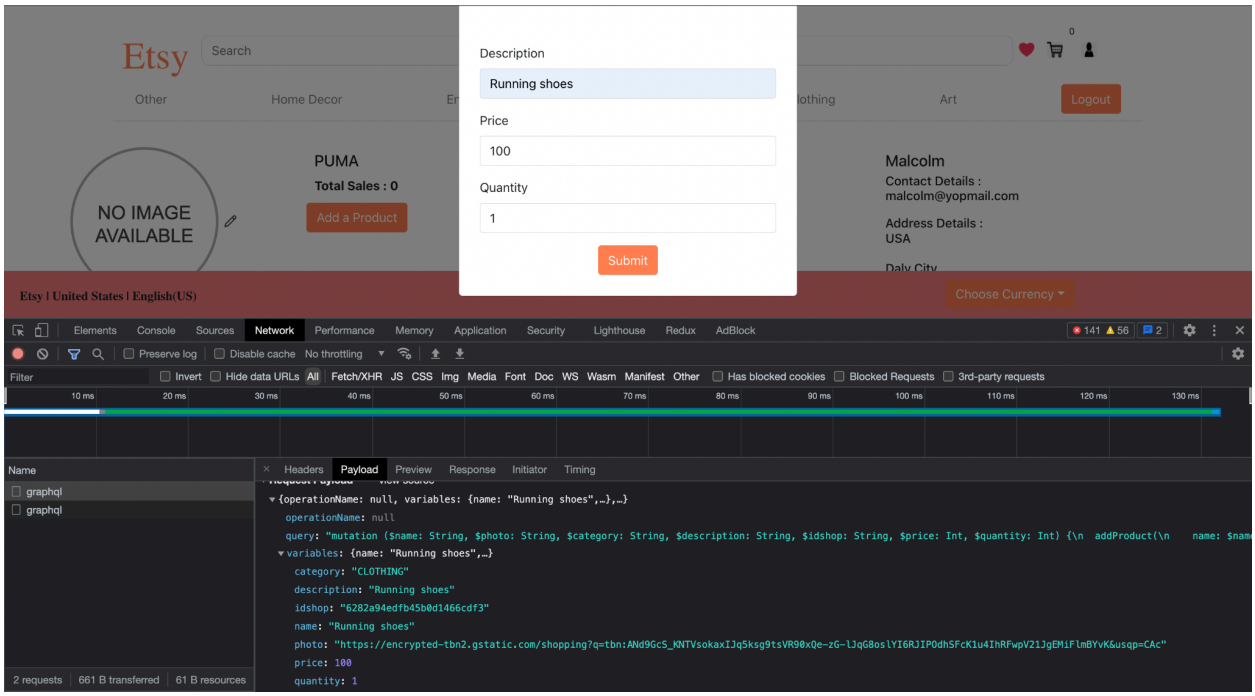


4. Show the entry for Malcolm in your database and highlight the change in the city. (MySQL or MongoDB)



5. Create a shop “Puma” and add an item “Running Shoe” with quantity as 1.

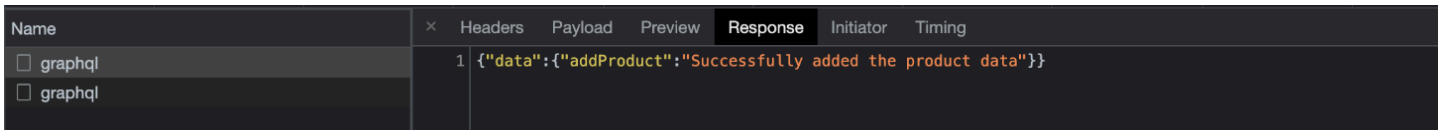




6. Show the query at the backend adding the data for “Running Shoe”.10. Show the query at the backend adding the data for “Running Shoe”.

```
addProduct:{
    type: GraphQLString,
    args: {
        name: { type: GraphQLString },
        photo: { type: GraphQLString },
        category:{type: GraphQLString },
        description: { type: GraphQLString },
        price:{type:GraphQLInt},
        quantity:{type:GraphQLInt},
        idshop: { type: GraphQLString }
    },
    async resolve(parent, args) {
        try {
            product = new Product({
            name:args.name,
            photo:args.photo,
            category:args.category,
            description:args.description,
            price:args.price,
            quantity:args.quantity,
            idshop:args.idshop,
            totalSales: 0
            })
            await product.save()
            return("Successfully added the product data");
        } catch (err) {
            console.log("Added product: ", err);
            return("Added product")
        }
    }
},
```

-



7. Show the frontend query file (javascript file) comprising all the queries.

```
import { gql } from 'apollo-boost';

export const getAllProducts = gql`
    query {
        getAllProducts{
            _id
            name
            photo
            category
            description
            price
            quantity
            idshop
            totalSales
        }
    }
`;

export const getProductofShop = gql`
    query ($idshop:String){
        getProductofShop(
            id: $idshop,
        ){
        _id
        name
        photo
        category
        description
        price
        quantity
        idshop
        totalSales
    }
    }
`;

export const getProductbyId = gql`
    query ($idproduct:String){
        getProductbyId(
            id: $idproduct
        ){
        _id
        name
        photo
        category
        description
        price
        quantity
        idshop
        totalSales
    }
    }
`;
```

8. Show the frontend mutation file (javascript file) comprising all the mutations.

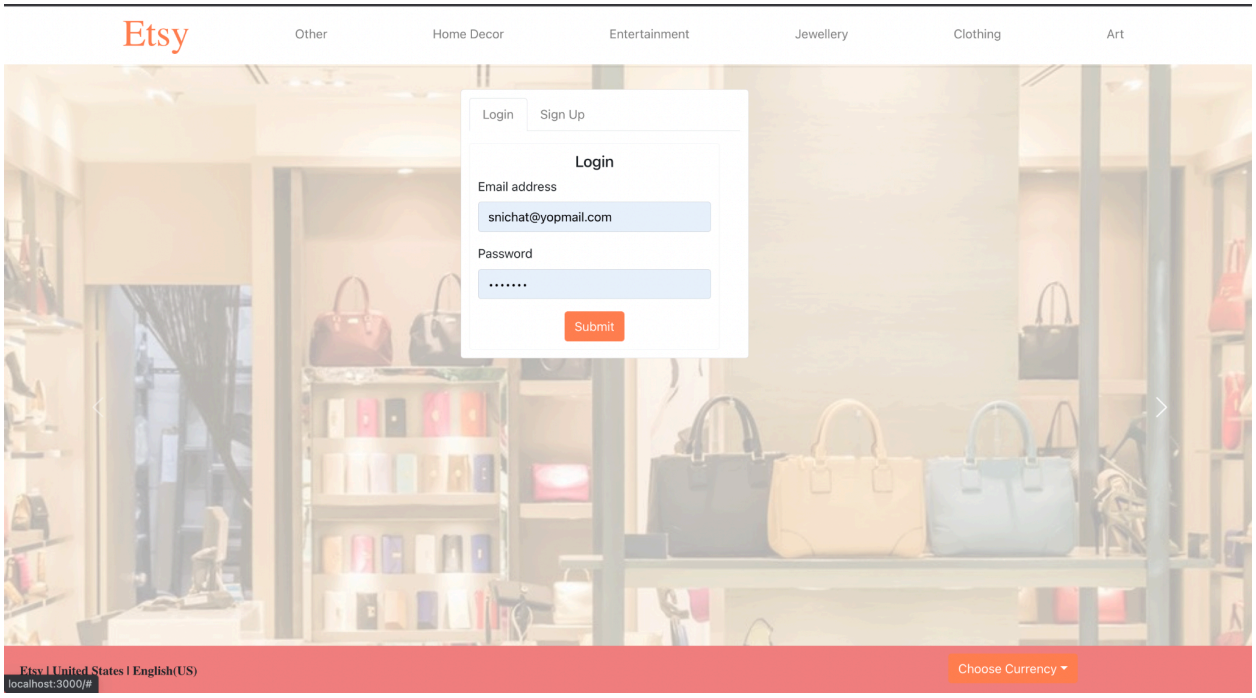
```
import { gql } from 'apollo-boost';

export const signupUser = gql`
    mutation ($name: String, $email: String, $password: String){
        signupUser(name: $name, email: $email, password: $password){
            _id
            name
            email
            image
        }
    }
`;

export const updateUser = gql`
    mutation ($iduser:String ,$country: String, $city: String, $street: String, $image: String, $email: String){
        updateUser(
            iduser: $iduser,
            email: $email,
            image: $image,
            city: $city,
            street: $street,
            country: $country,
            pinCode: 0
    )
    }
`;

export const addProduct = gql`
    mutation ($name:String ,$photo: String, $category: String, $description: String, $idshop: String, $price: Int, $quantity: Int){
        addProduct(
            name: $name,
            photo: $photo,
            category:$category,
            description: $description,
            price: $price,
            quantity:$quantity,
            idshop: $idshop
    )
    }
`;

export const addOrders = gql`
    mutation ($iduser:String ,$productItems: [OrderProducInputtType!]){
        addOrders(
            iduser: $iduser,
            productItems: $productItems
    )
    }
`;
```

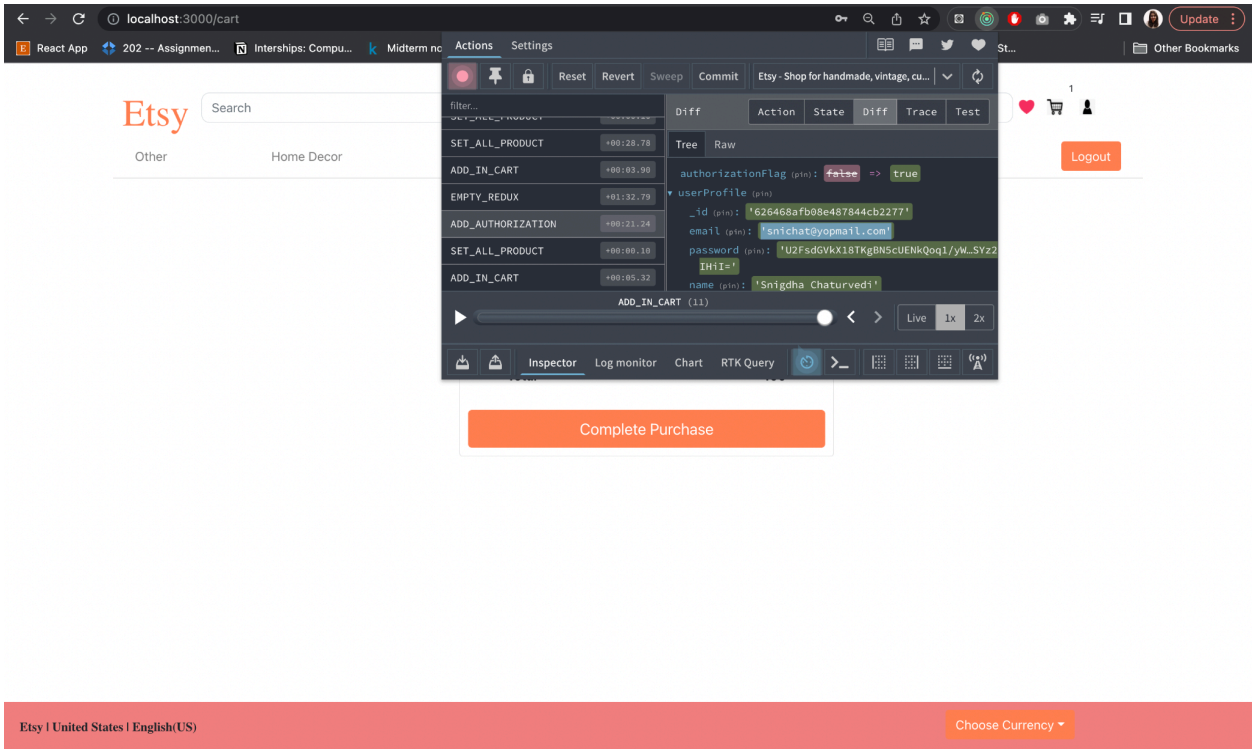
9. Show the backend schema where the GraphQL objects are created.

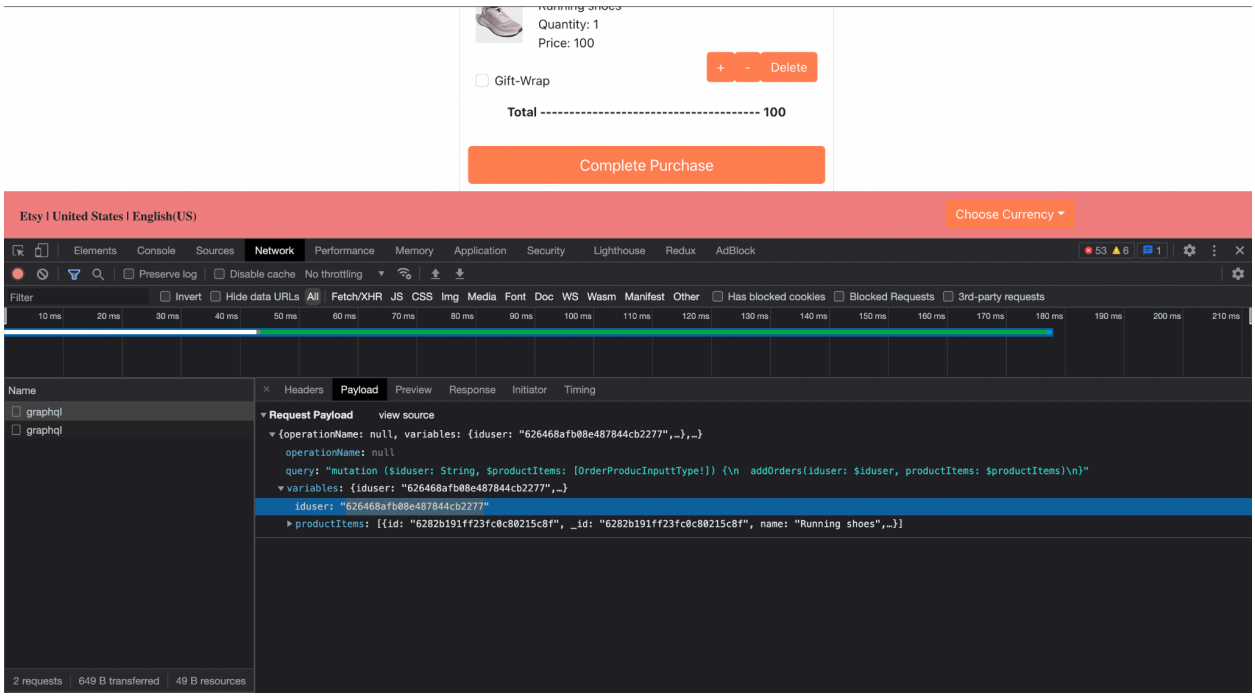
```
const ShopType = new GraphQLObjectType({
    name: 'ShopType',
    fields: () => ({
        _id: { type: GraphQLID },
        name: { type: GraphQLString },
        image: { type: GraphQLString },
        iduser: { type: GraphQLString },
        totalSales:{type:GraphQLInt},
        // productIds:{type:GraphQLList}
    })
});

const ProductType = new GraphQLObjectType({
    name: 'ProductType',
    fields: () => ({
        _id: { type: GraphQLID },
        name: { type: GraphQLString },
        photo: { type: GraphQLString },
        category:{type: GraphQLString },
        description: { type: GraphQLString },
        price:{type:GraphQLInt},
        quantity:{type:GraphQLInt},
        idshop: { type: GraphQLString },
        totalSales:{type:GraphQLInt}
    })
});

const OrderProducInputtType = new GraphQLInputObjectType({
    name: 'OrderProducInputtType',
    fields: () => ({
        name: { type: GraphQLString },
        category: { type: GraphQLString },
        photo: { type: GraphQLString },
        purchasedPrice:{type:GraphQLInt},
        purchasedQuantity:{type:GraphQLInt},
        idshop: { type: GraphQLString }
    })
})
```

```
const OrderProductType = new GraphQLObjectType({
    name: 'OrderProductType',
    fields: () => ({
        name: { type: GraphQLString },
        category: { type: GraphQLString },
        photo: { type: GraphQLString },
        purchasedPrice:{type:GraphQLInt},
        purchasedQuantity:{type:GraphQLInt},
        idshop: { type: GraphQLString }
    })
})

const AddressType = new GraphQLObjectType({
    name: 'AddressType',
    fields: () => ({
        street: { type: GraphQLString },
        city: { type: GraphQLString },
        country: { type: GraphQLString },
        pinCode:{type:GraphQLInt}
    })
})

const UserType = new GraphQLObjectType({
    name: 'UserType',
    fields: () => ({
        _id: { type: GraphQLString },
        email: { type: GraphQLString },
        password: { type: GraphQLString },
        name: { type: GraphQLString },
        image:{type:GraphQLString},
        // dateOfBirth:{type:GraphQLDate},
        phoneNum: { type: GraphQLInt },
        address:{type:AddressType},
        favourites:{type:new GraphQLList(GraphQLString)}
    })
})
const OrderType = new GraphQLObjectType({
    name: 'OrderType',
    fields: () => ({
        _id: { type: GraphQLID },
        iduser: { type: GraphQLString },
        productItems: { type:new GraphQLList(OrderProductType)}
    })
});
```

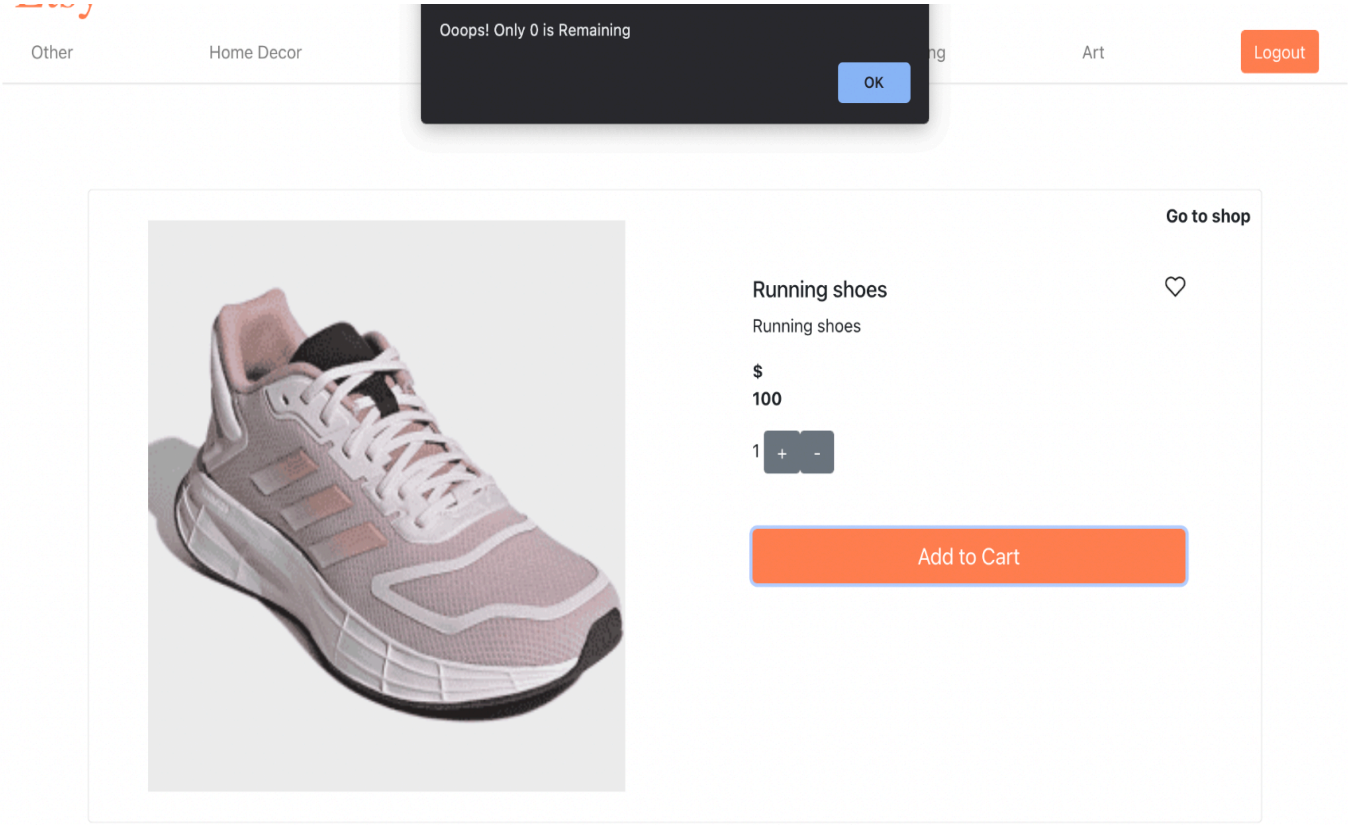
10.Create an order for “Running Shoe” from **any other user's** account.






Quantity: 1
Price: 100
+
-
Delete
Gift-Wrap
Total ---------- 100
Complete Purchase
Etsy | United States | English(US)
Choose Currency
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Redux
AdBlock
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
10 ms
20 ms
30 ms
40 ms
50 ms
60 ms
70 ms
80 ms
90 ms
100 ms
110 ms
120 ms
130 ms
140 ms
150 ms
160 ms
170 ms
180 ms
190 ms
200 ms
210 ms
Name
graphql
graphql
Headers
Payload
Preview
Response
Initiator
Timing
Request Payload
view source
{operationName: null, variables: {iduser: "626468afb08e487844cb2277",…},…}
operationName: null
query: "mutation ($iduser: String, $productItems: [OrderProducInputtType!]) {\n addOrders(iduser: $iduser, productItems: $productItems)\n}"
variables: {iduser: "626468afb08e487844cb2277",…}
iduser: "626468afb08e487844cb2277"
productItems: [{id: "6282b191ff23fc0c80215c8f", _id: "6282b191ff23fc0c80215c8f", name: "Running shoes",…}]
2 requests
649 B transferred
49 B resources

11. Open the product “Running Shoe” and highlight the quantity.



Questions

1. How will you enable multi-part data in GraphQL?

   The files from the multipart-request have to be processed with multer which makes a local copy of the image . Which is uploaded to S3 that returns a public URL of the image. After that, the files can be processed by the resolver function. We use Apollo-client on the client side to process GraphQL requests in the application. Hence, we need to create our own Network Interface that can handle multipart queries.

2. Discuss the architecture for using multi-part data in GraphQL without using any open source library from Git?

   We can store the image on S3 and the public URL generated can be stored in the database via a mutation.

3. State any open source library for enabling multi-part data transfer using GraphQL with sample code. Argue why do you think that this particular library is a good fit?

   Apollo-upload-server is a GraphQL library for file uploads. It is very easy to use and GraphQL has provided really clean and easy to understand documentation for usage. It is quick, easy and scales well.

   ```
   import { apolloUploadExpress as AUE } from 'apollo-upload-server';
   app.use( '/graphql', bodyParser.json(),
   AUE(/* Options */),
   graphqlExpress(/* … */))
   ```

   Alternatively react-s3 is a library that can assist the frontend to upload the images directly to the s3 bucket and return a public url that can be sent back to the backend for being stored.

```
const region = "us-east-X"
const bucketName = "etsyXXXXXXXXX"
const accessKeyId = "AKIXXXXXXXXXXXXXXXXX"
const secretAccessKey = "OzT5CXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

const config ={
  region:region,
  bucketName:bucketName,
  accessKeyId:accessKeyId,
  secretAccessKey:secretAccessKey,
  dirName:""
}
```

```
const onImageUpload= async(file)=>{
return new Promise(resolve => {
 const reader = new FileReader();
 S3FileUpload.uploadFile(file,config).then(data=>{
   var strc = data.location.slice(0,locat+3)+region+"."+data.location.slice(locat+3,)
   console.log(data.location.slice(0,locat+3),region,data.location.slice(locat+3,))
   resolve(strc)
})})}
```