EBS

EASY BUY/ SELL

Team: 6

Vidya Damalacheruvu
Karthik Yanagandula
Gayathri Garikapati
Pujitha Talluri

Story of the project:

We make it possible for millions of people around the country to buy and sell practically anything. We all have items that we never use, haven't used in a long time, or have just outgrown. But, because these items still have worth, why not sell them, or donate them to people in need? We've proposed a project called Easy Buy and Sell, in which a client may sell and buy things. Based on the items mentioned, we will categorize the products. This project makes it much easy to trade stuff than any other. We are also attempting to incorporate split delivery and payment choices in addition to these features.

Technologies used:

We have used MEAN Stack technologies such as MongoDB, Express, Angular, NodeJS for building this Ecommerce website.

Front-end: HTML, CSS, Bootstrap, Angular
Back-end: NodeJS, Express
Database: MongoDB Cloud
APIs Used: Algolia Search, Stripe Payment

Data details:

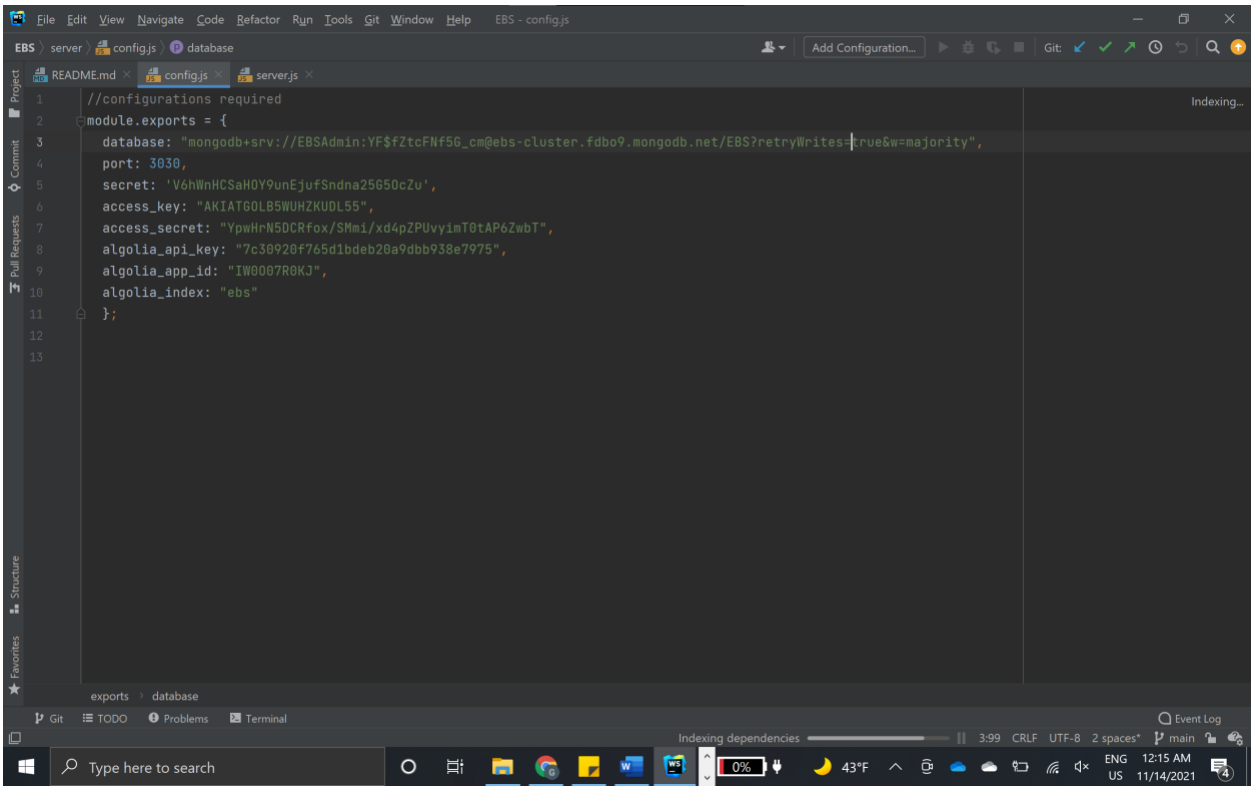
For any Ecommerce website we need,

1. User data
2. Product data
3. Categories data
4. Orders data
5. Reviews data

For this project, we are using MongoDB hosted on cloud and stores data. Stored the product images in AWS S3 bucket.

Algolia Search API is used to incorporate the product's search results into the project. To access this API, we need to have the API key and API id.

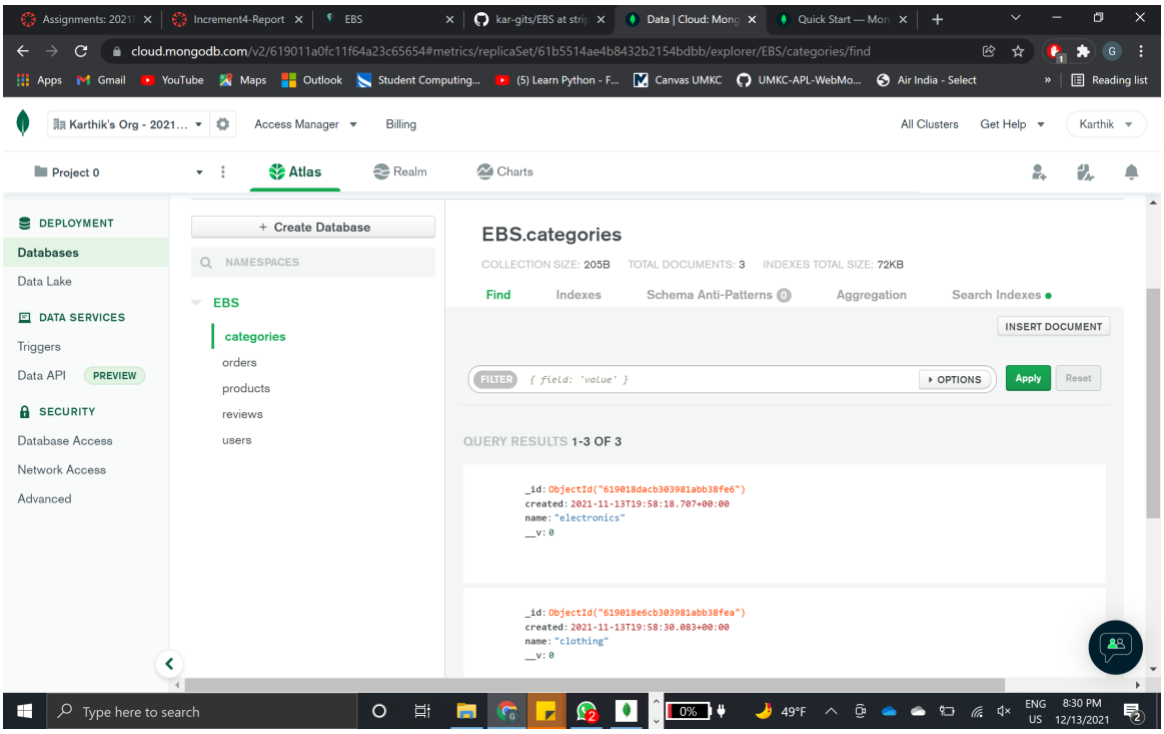
The Algolia API key and id, access key and secret for accessing AWS, MongoDB database URL and port are provided in the config.js file as shown below.



We are using Stripe API for payment options which facilitates the option to accept the online payments across the world.

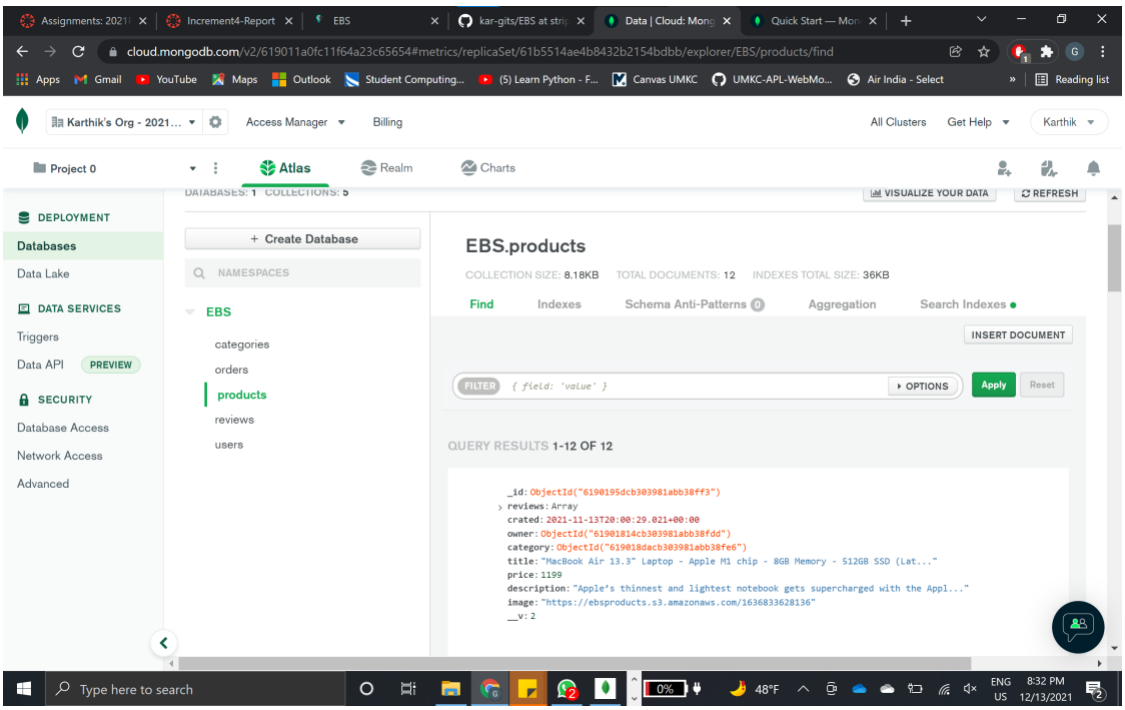
All the data corresponding to categories, products, users, orders, and reviews will be stored in MongoDB cloud.

Categories: With schema containing the details of Id, created date, category name**.**

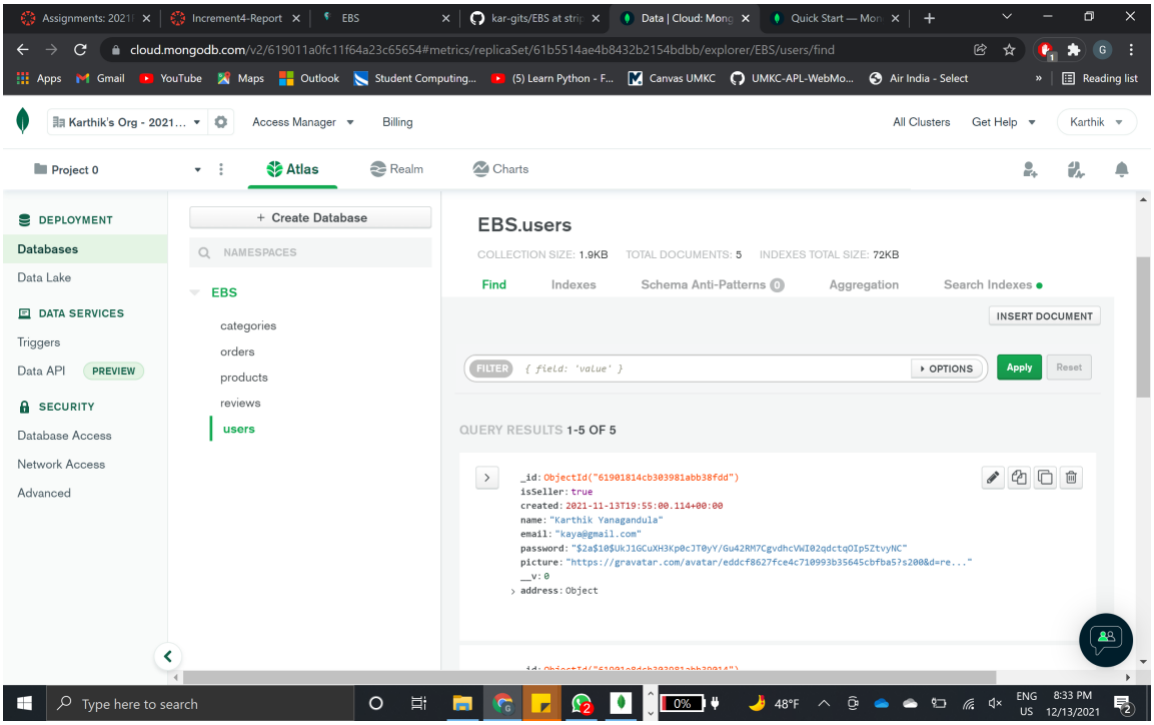


Products:

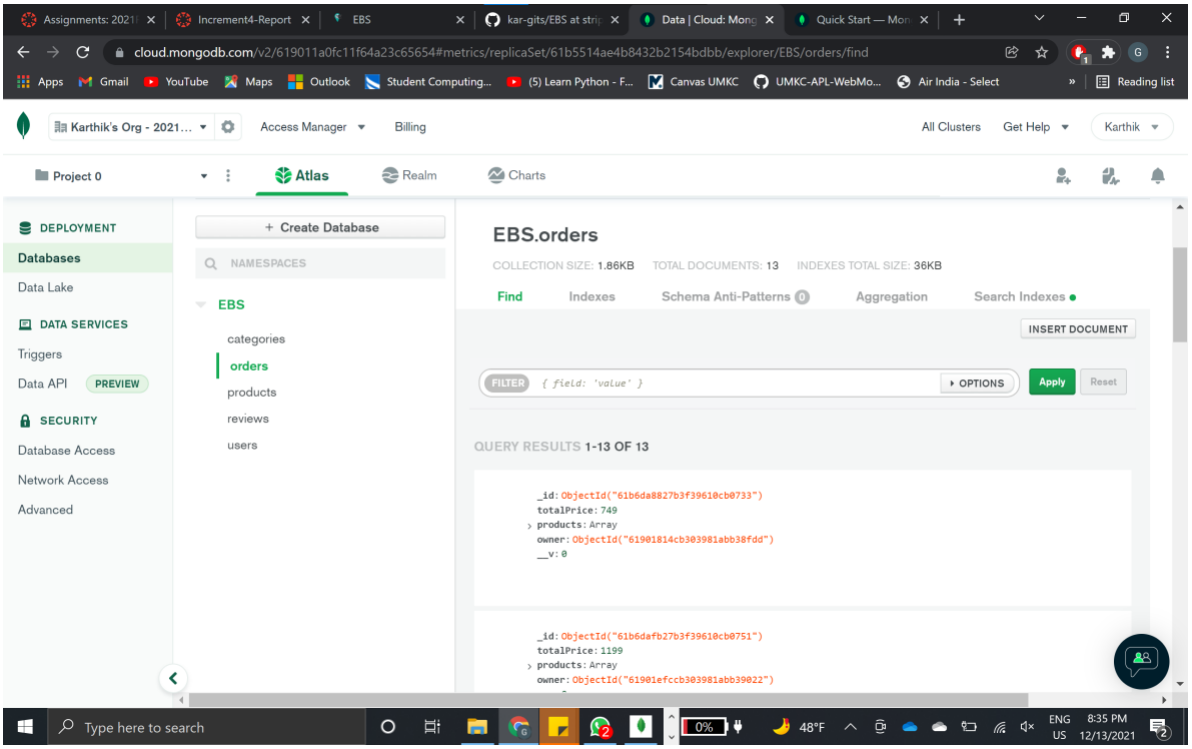
With products details containing schema contents as id, reviews, creates, owner, category, title, price, description, image.



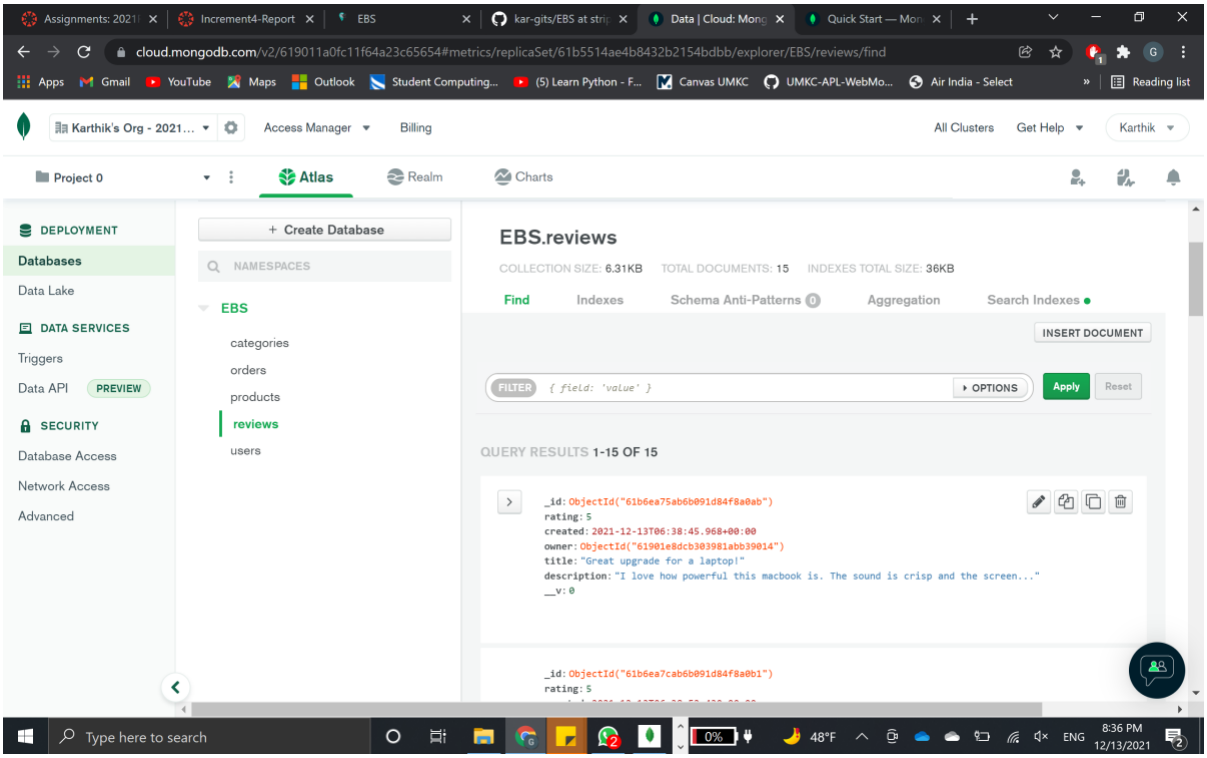
Users: Schema contains id, isSeller, created, name, email, password, picture, address.



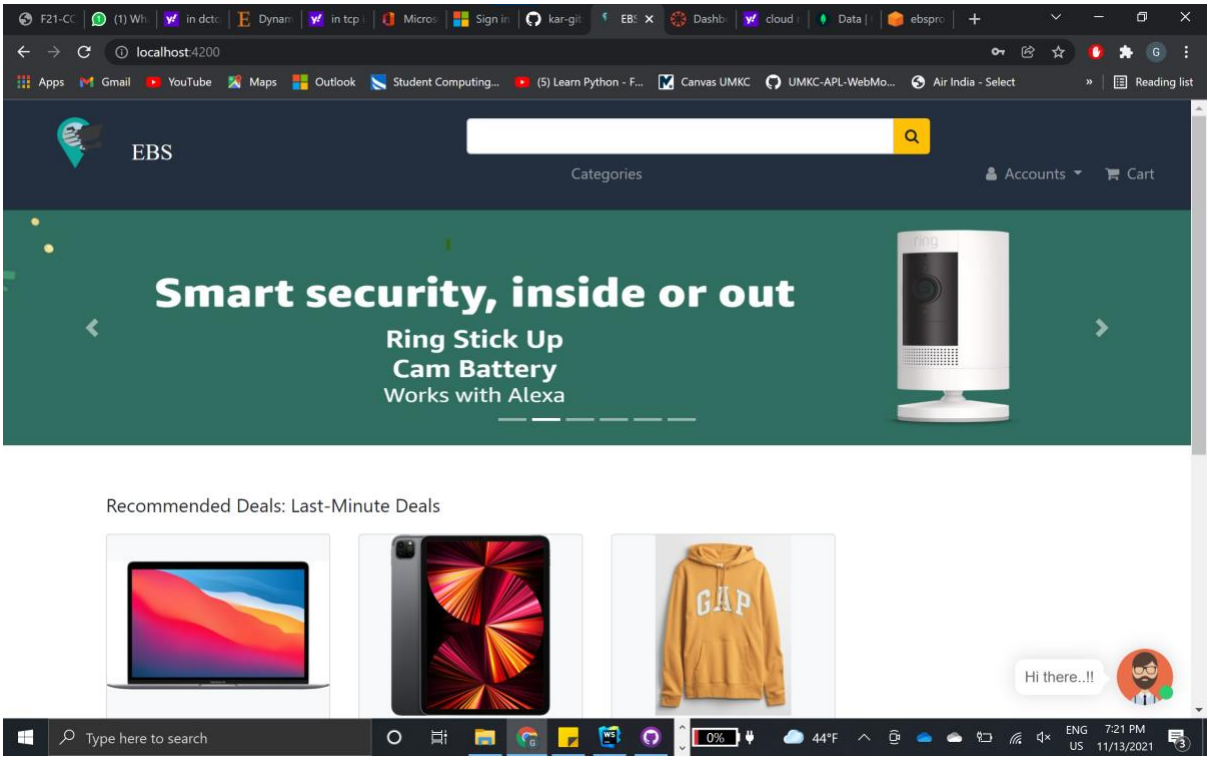
Orders: Schema contains id, total price, products, owner.



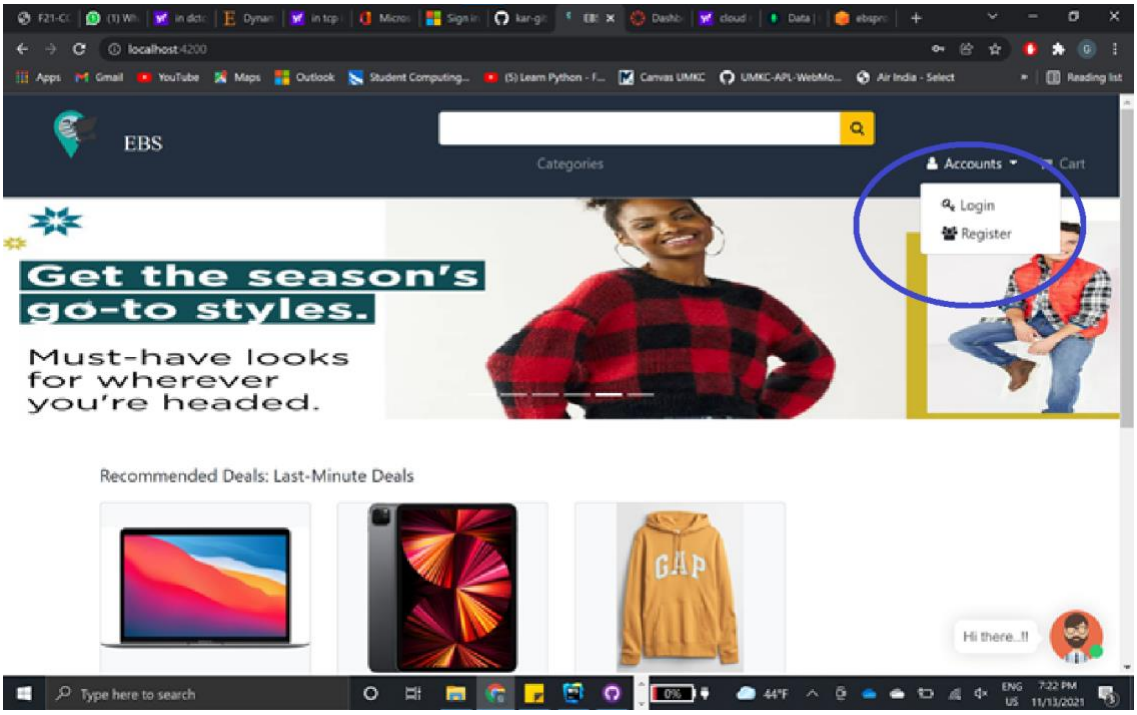
Reviews: Schema contains id, rating, created, owner, title, description.



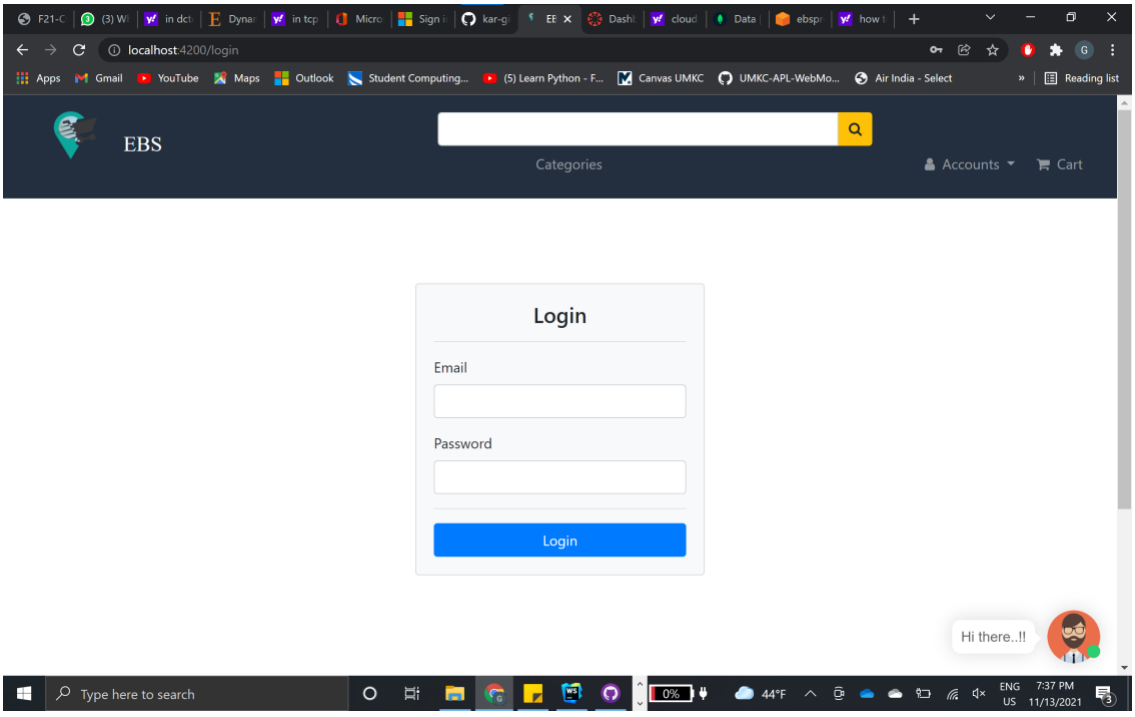
Working screens from project: Home screen of EBS application.



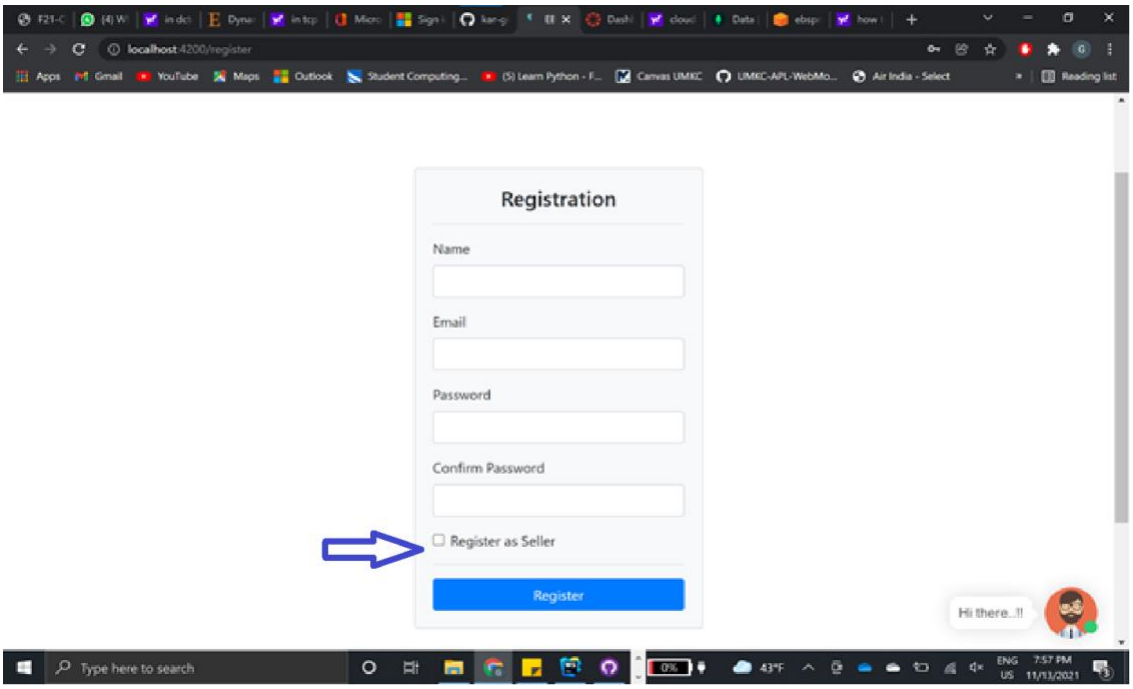
Here, we have two options under accounts i.e., Login and Register.



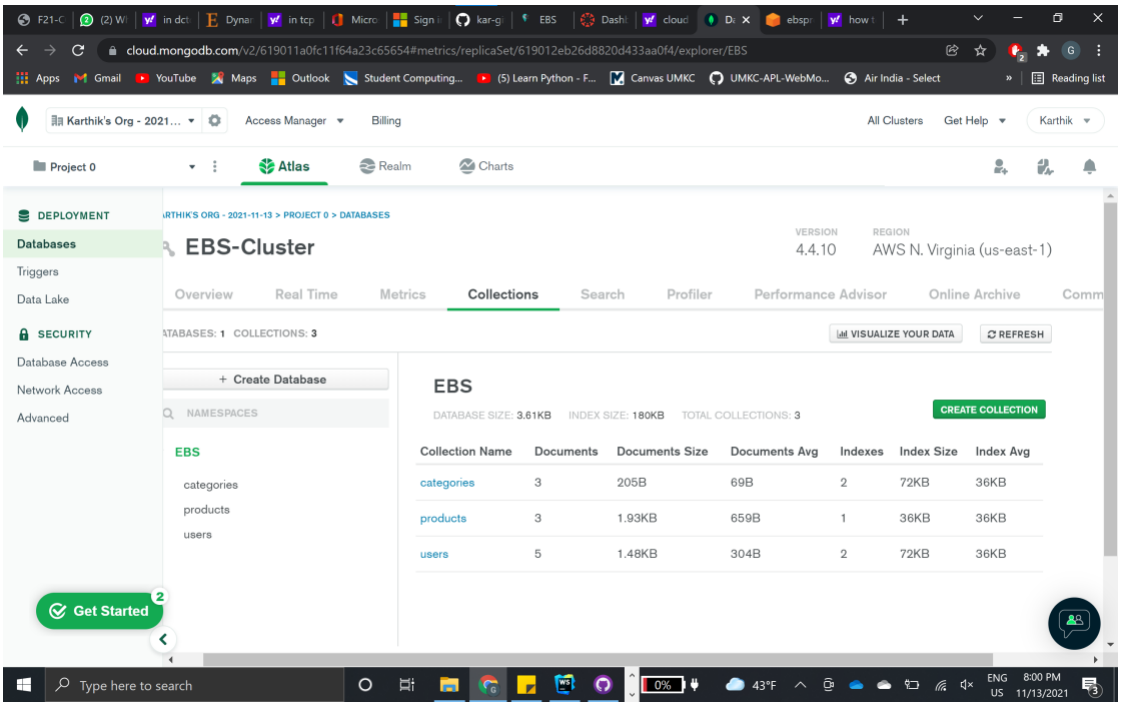
If it's an existing user, they can just click on login and provide the Email and password to access the application.



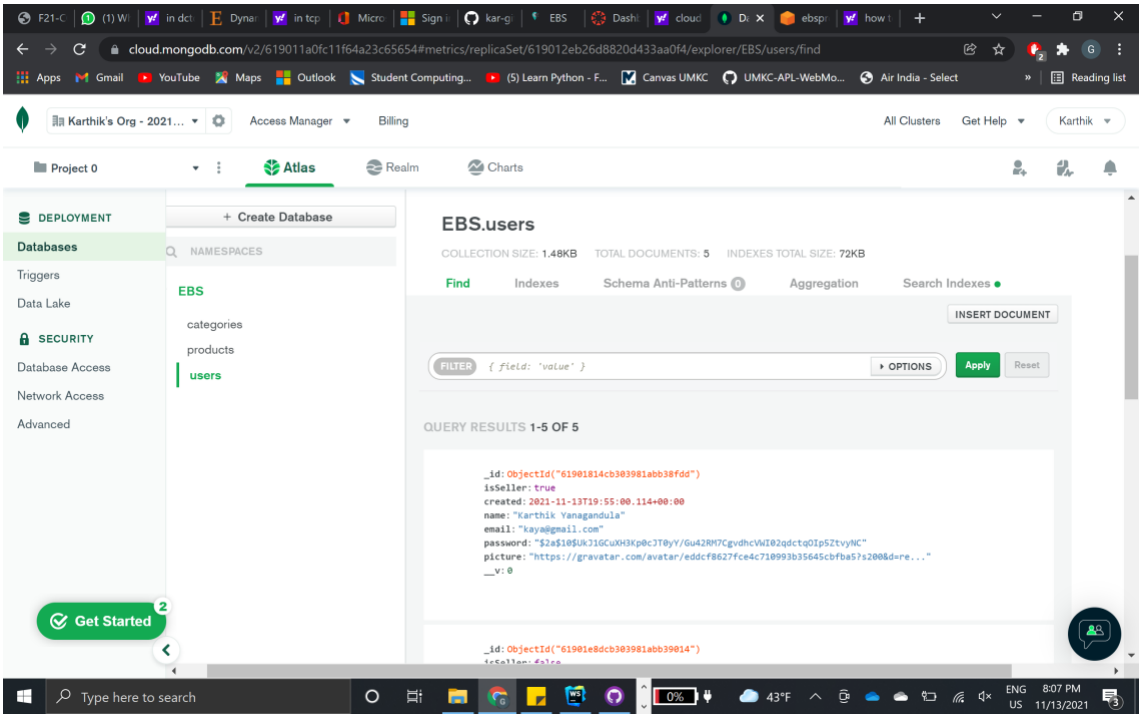
If the user is new to the application, they can click on register tab and provide the details such as Name, Email, password. Here, we have an option to register as a seller if the user want's their products under sale.



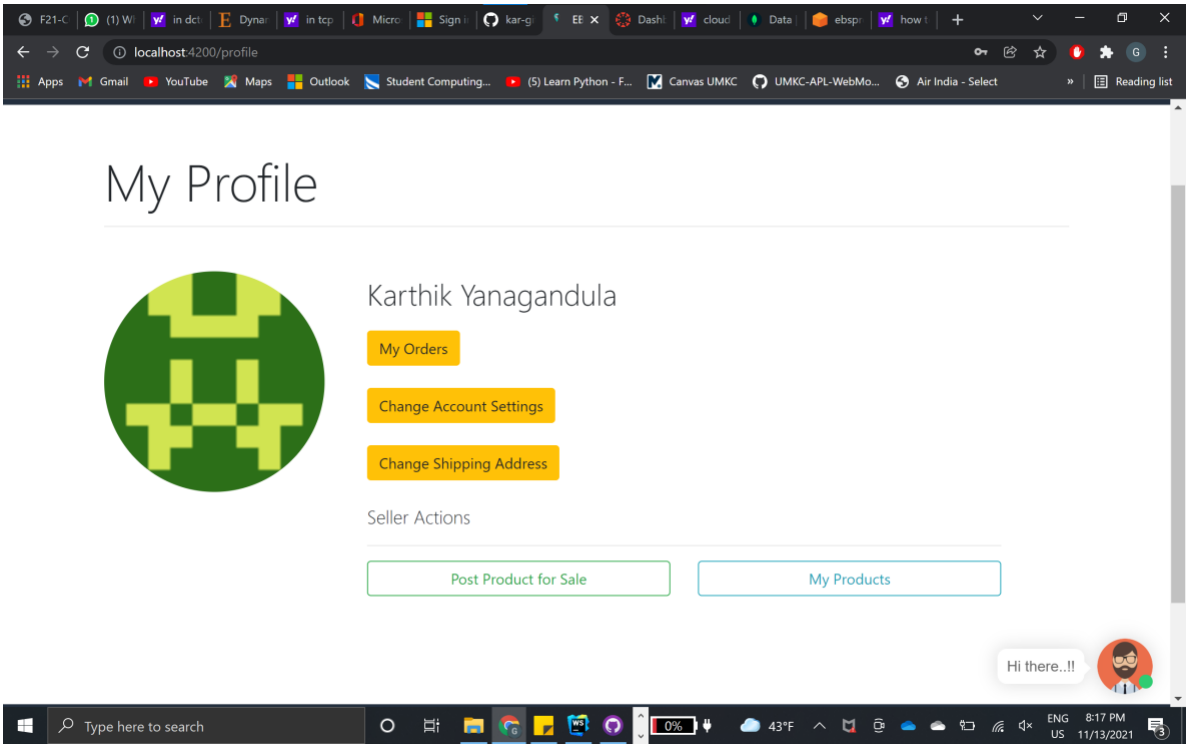
All this user data is stored in MongoDB database under EBS-Cluster.



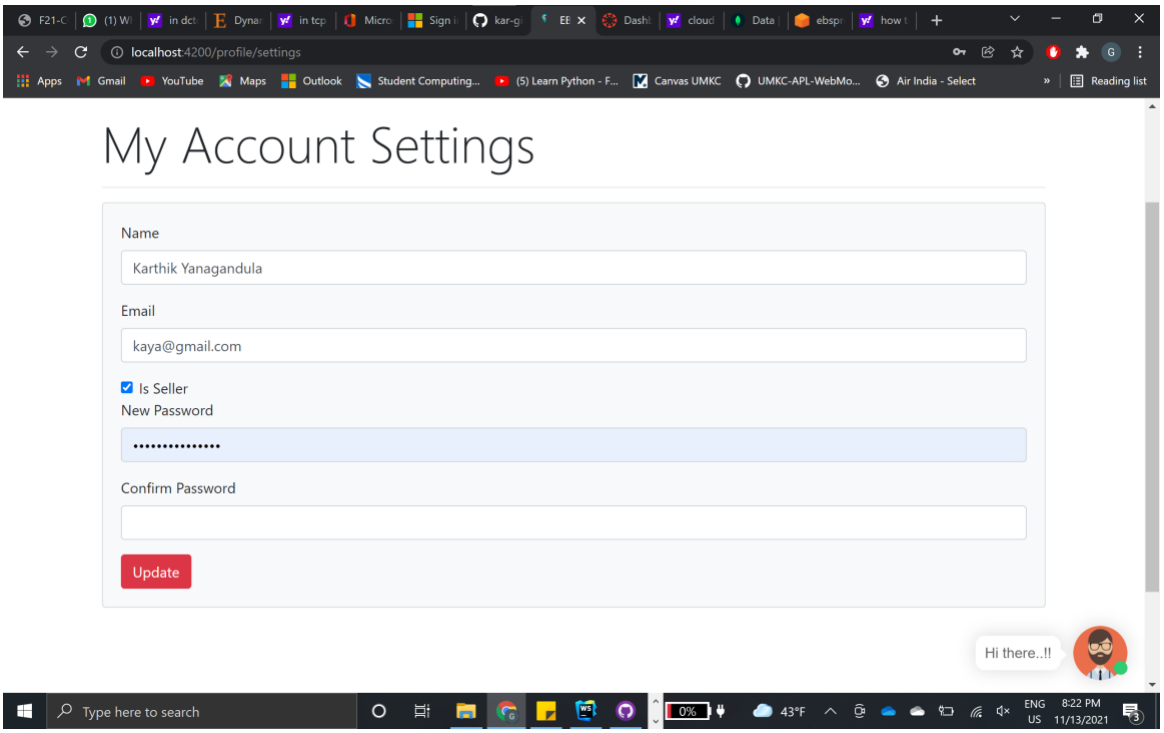
Below screen displays the schema for the user details.



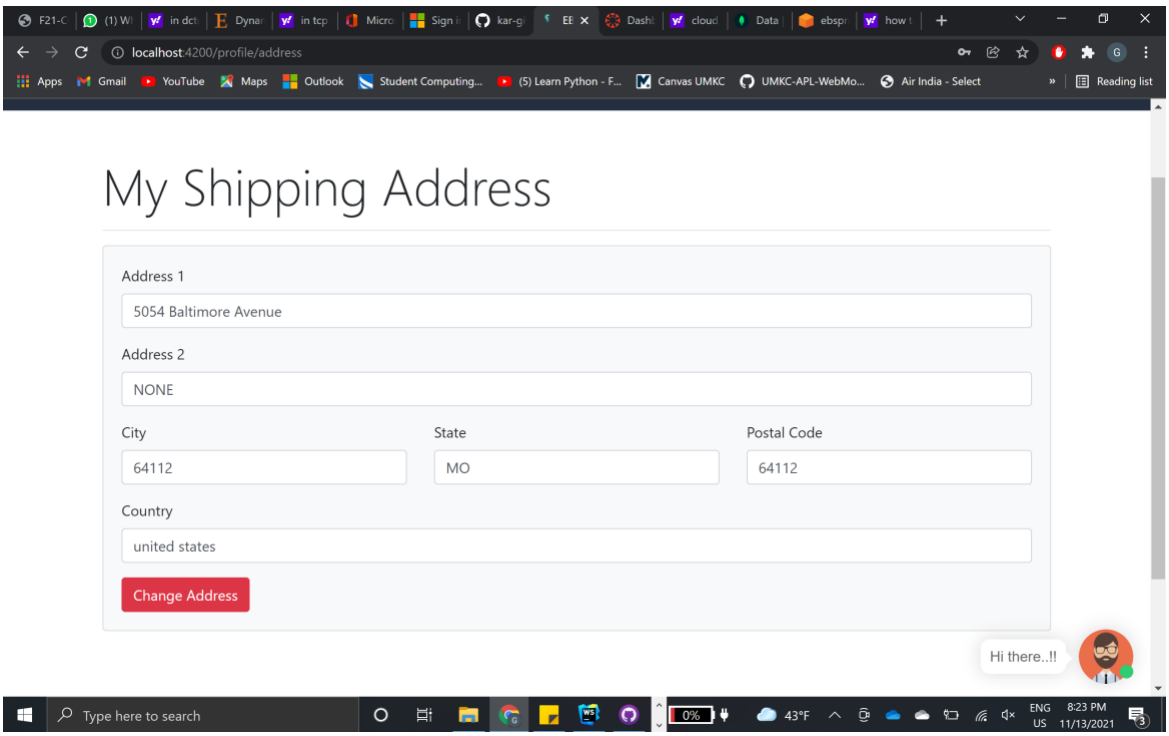
Below screen display the options for a user registered as a seller, he/she will have an option such as to post a product for sale and to view the products he has put for sale under my products tab.



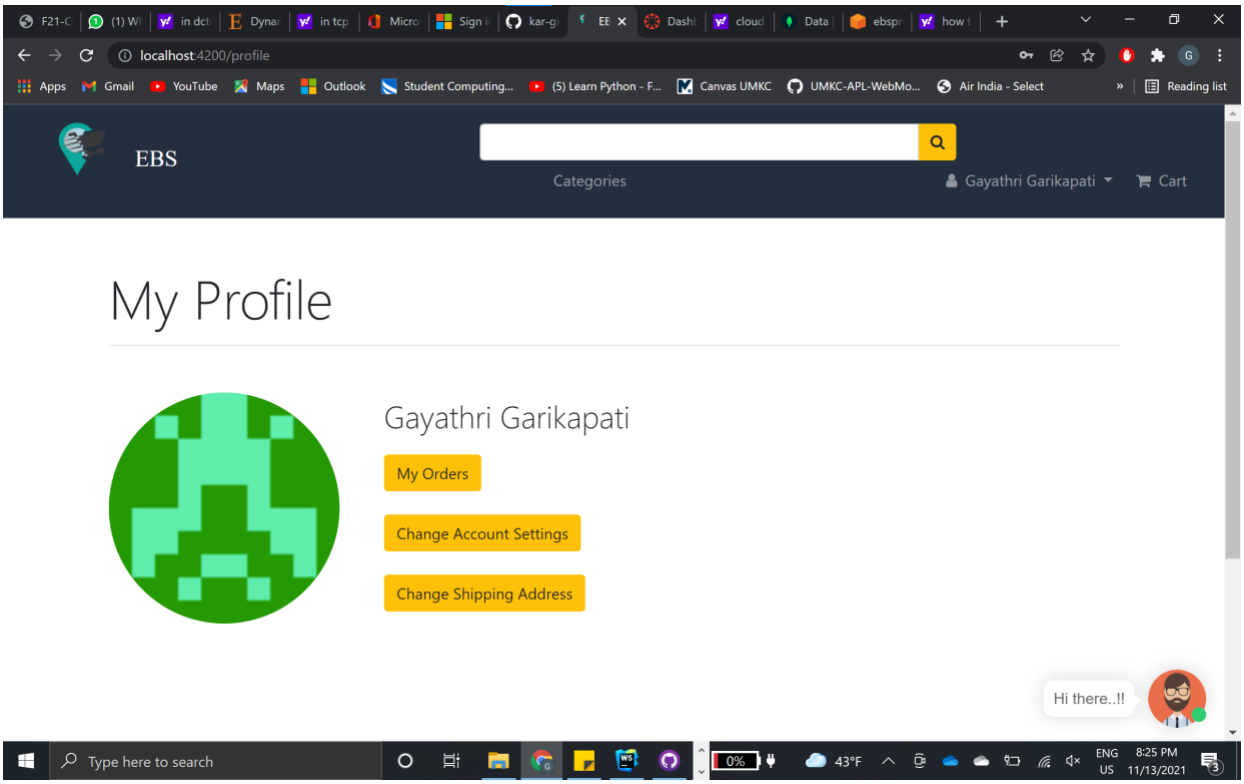
User has an option to change his account settings such as password update.



There is an option to update the shipping address.

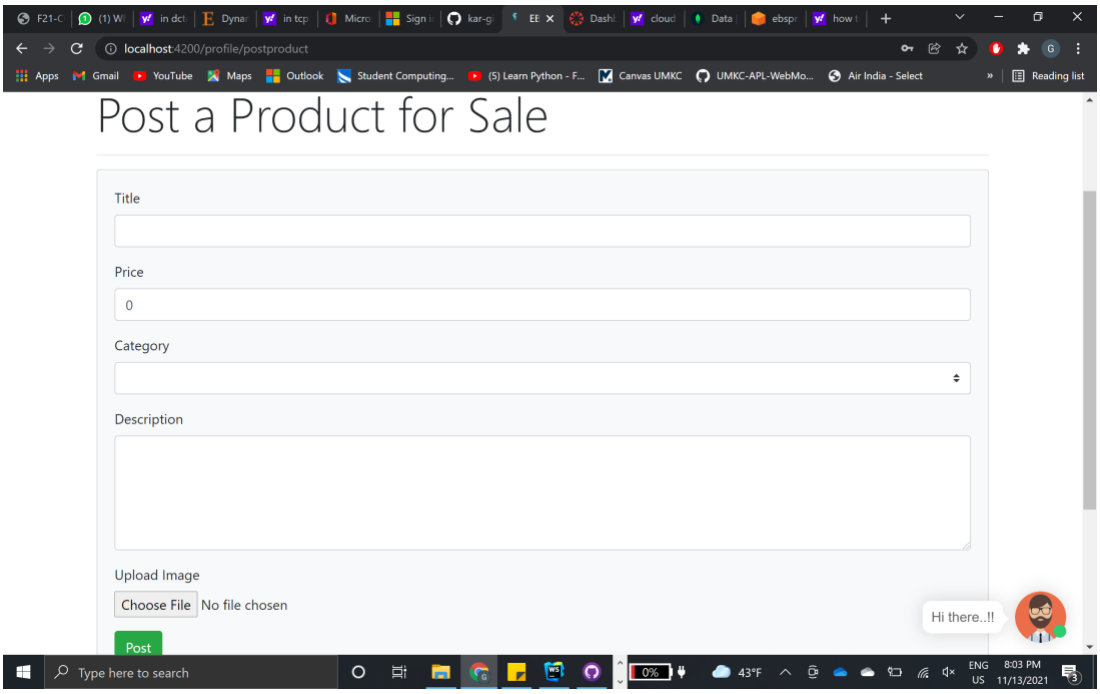


“Post product for sale” and “My products” options are not available for the user who is not registered as a seller.



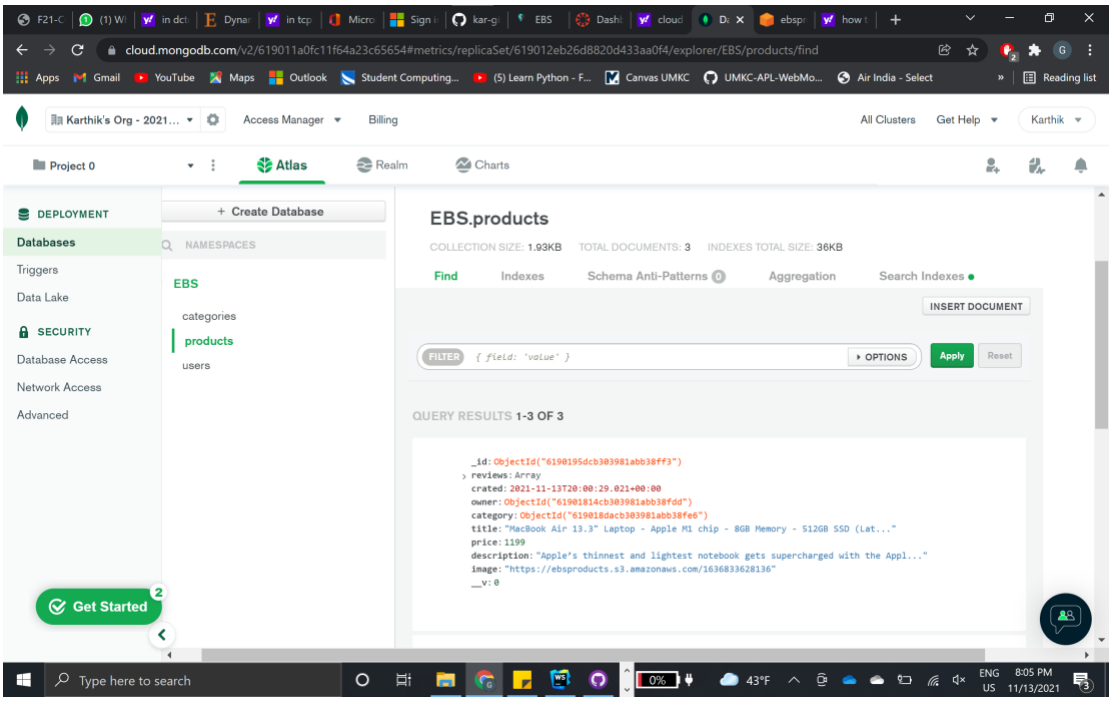
When the user wants to put a product for sale he can click on Post a product for sale tab and provide the details such as

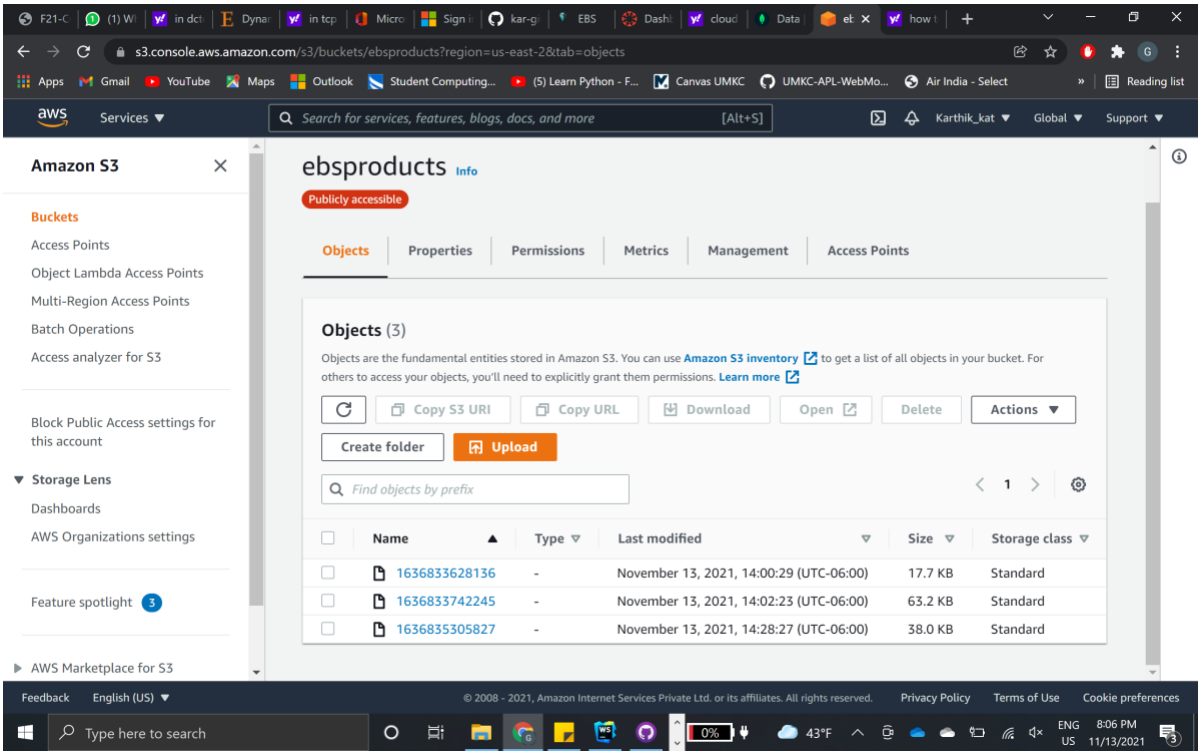
- Title of the product
- Price
- Category
- Description
- Upload the product image



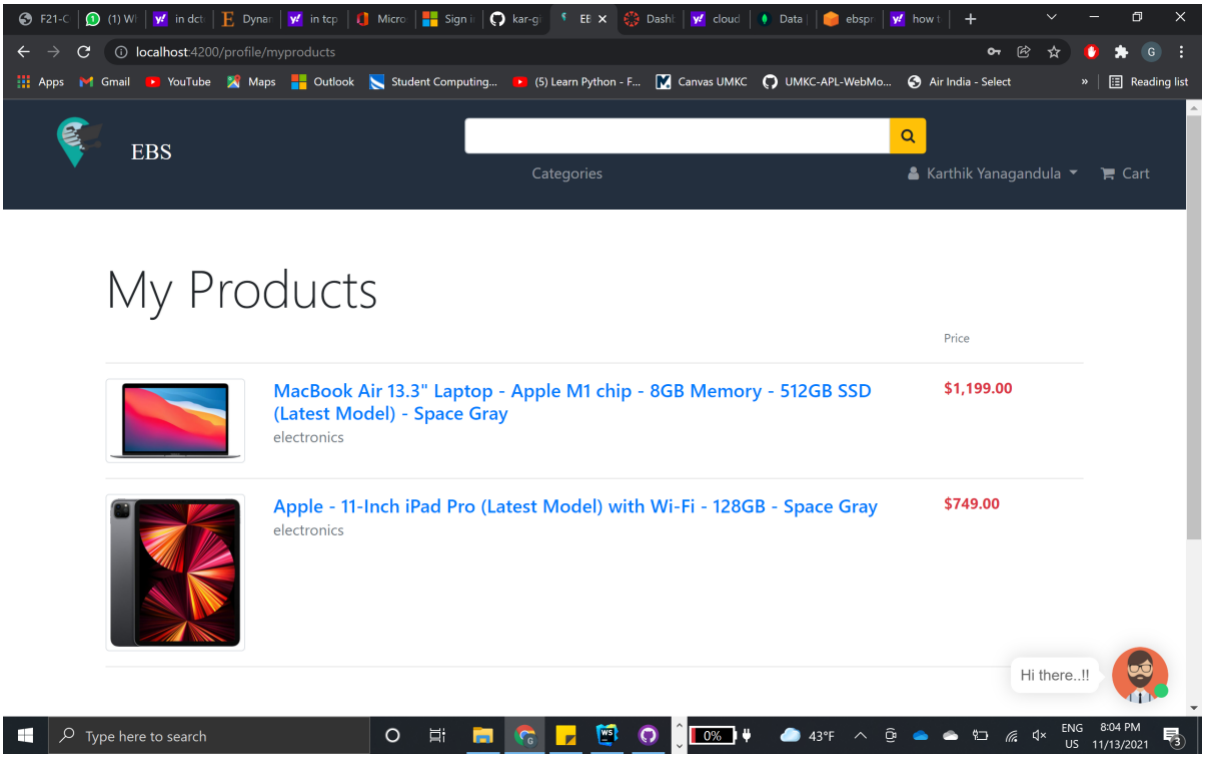
These products are stored in the “EBS-Cluster” of a MongoDB database hosted on cloud. All the product images are stored in AWS S3 bucket and we can see that the schema has AWS URL for the image value.

Here we have a product image with id 1636833628136 and this image is stored in the AWS S3 bucket.

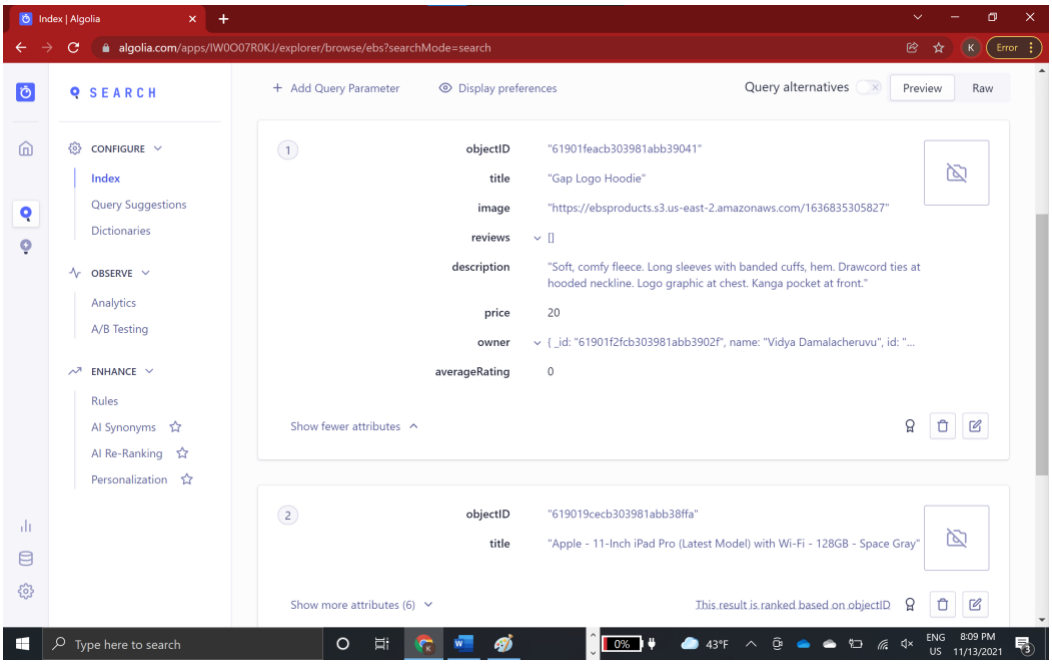




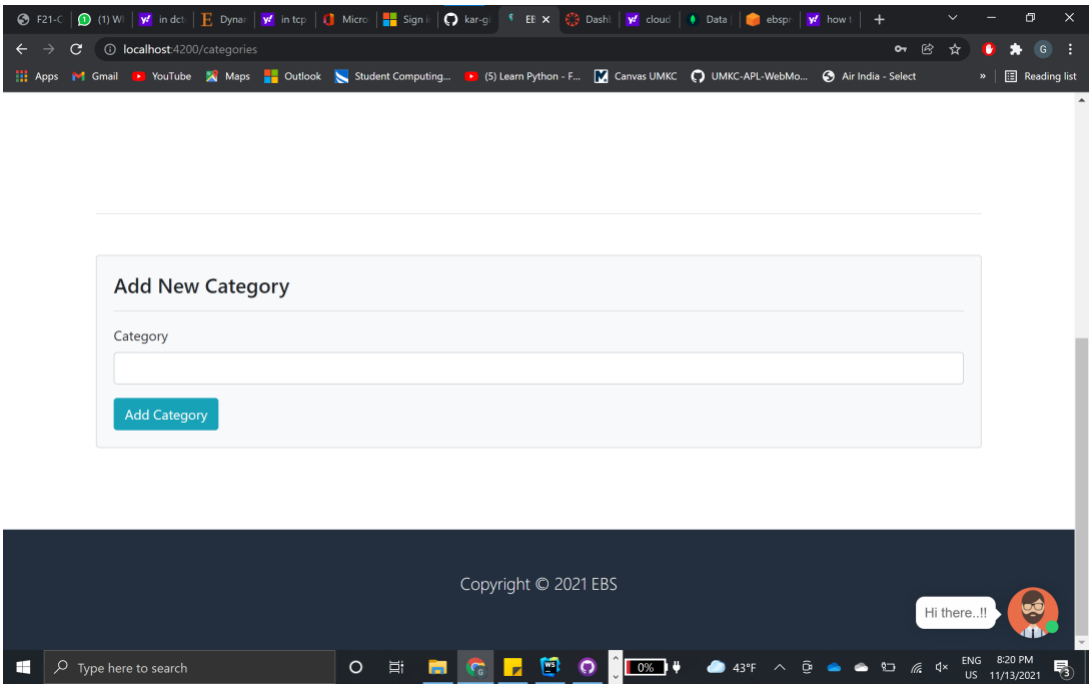
Under My Products tab all the products posted for sale by a particular user registered as a seller are visible.



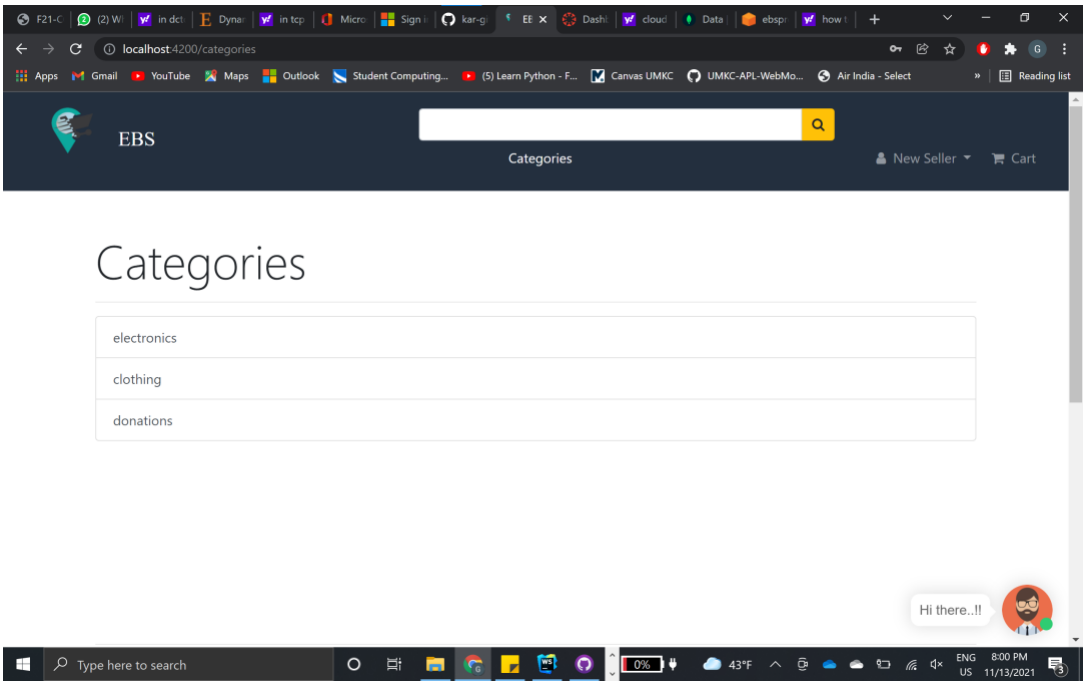
All these products are synchronized with Algolia Search API such that the product results can be pulled into the project.



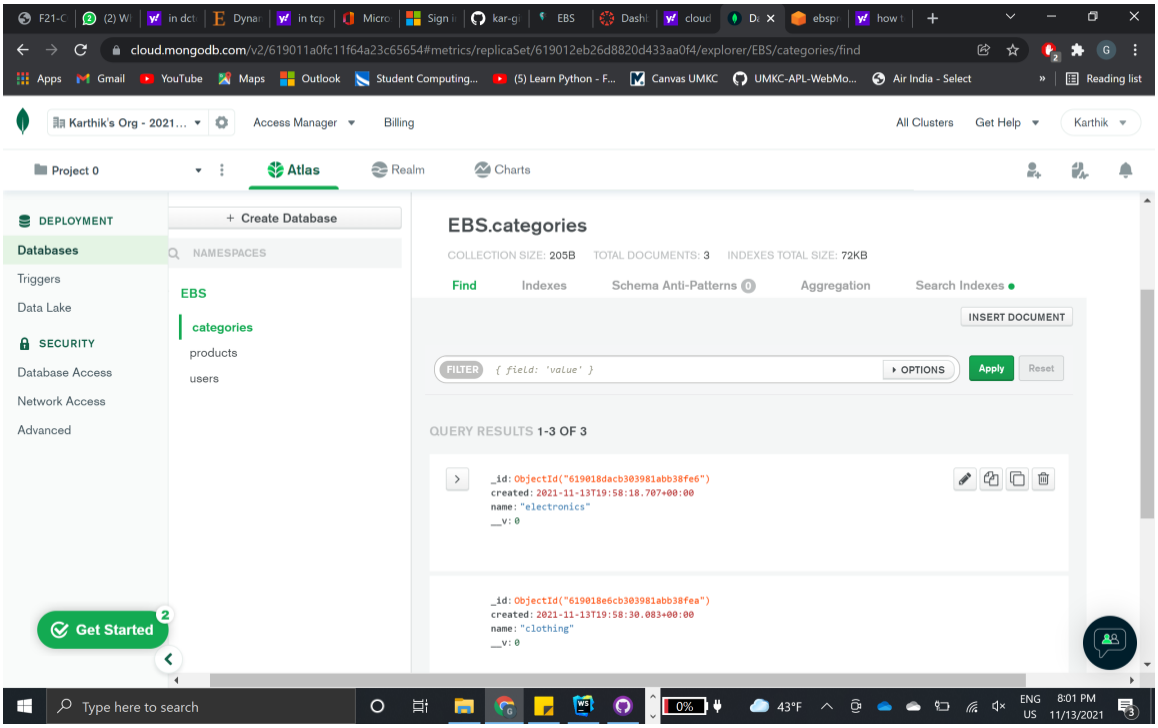
When the user is registered as a seller, he also has an option to add a new category of the product.



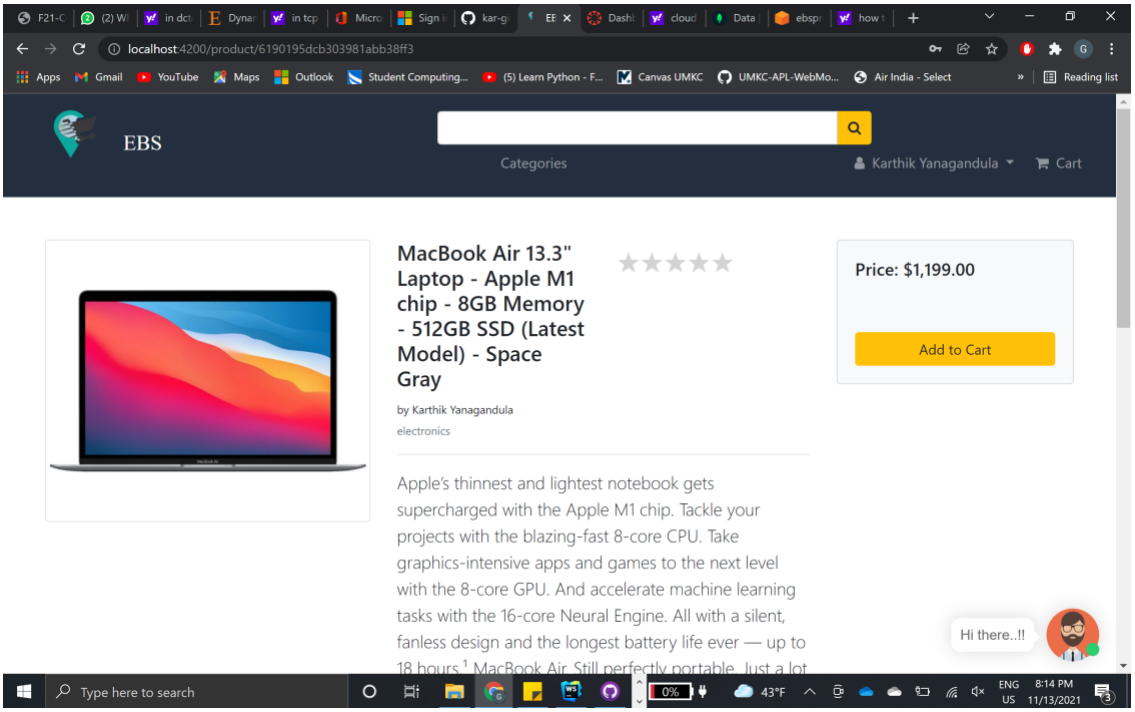
We have added the below categories as electronics, clothing, and donations for testing the application.



These categories are stored in MongoDB database under EBS-Cluster. Below screen displays the schema for a category.

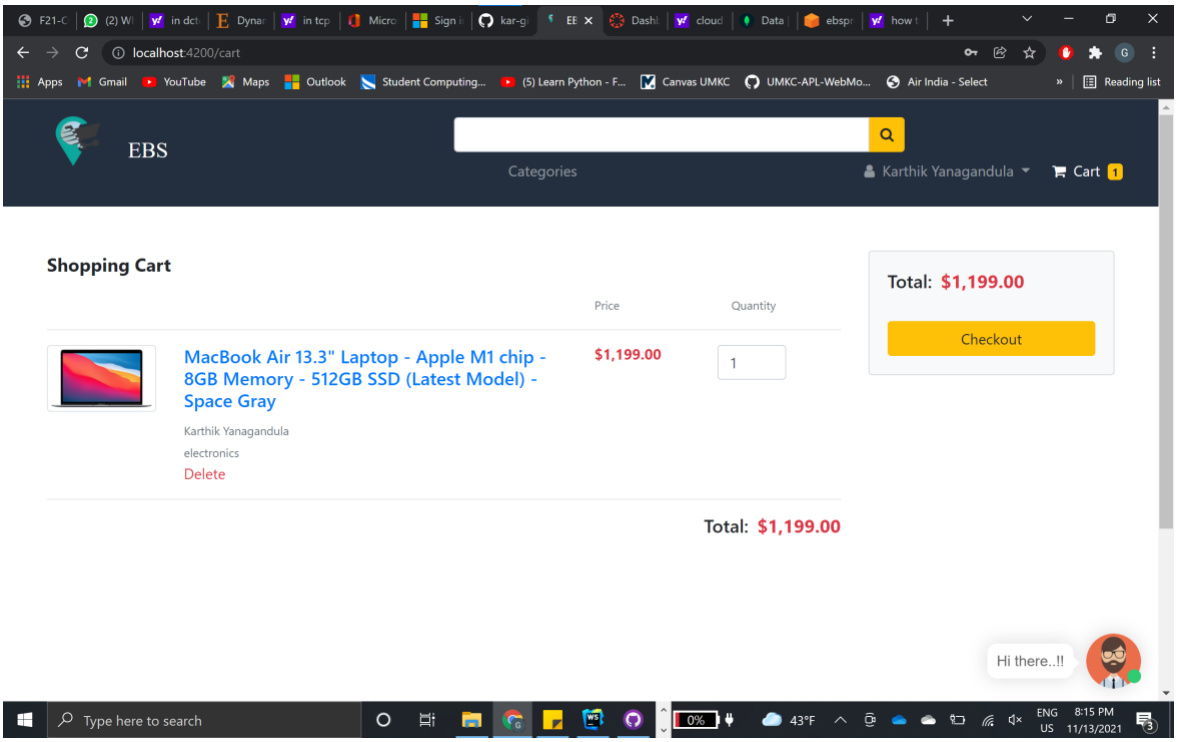


Add to cart option:



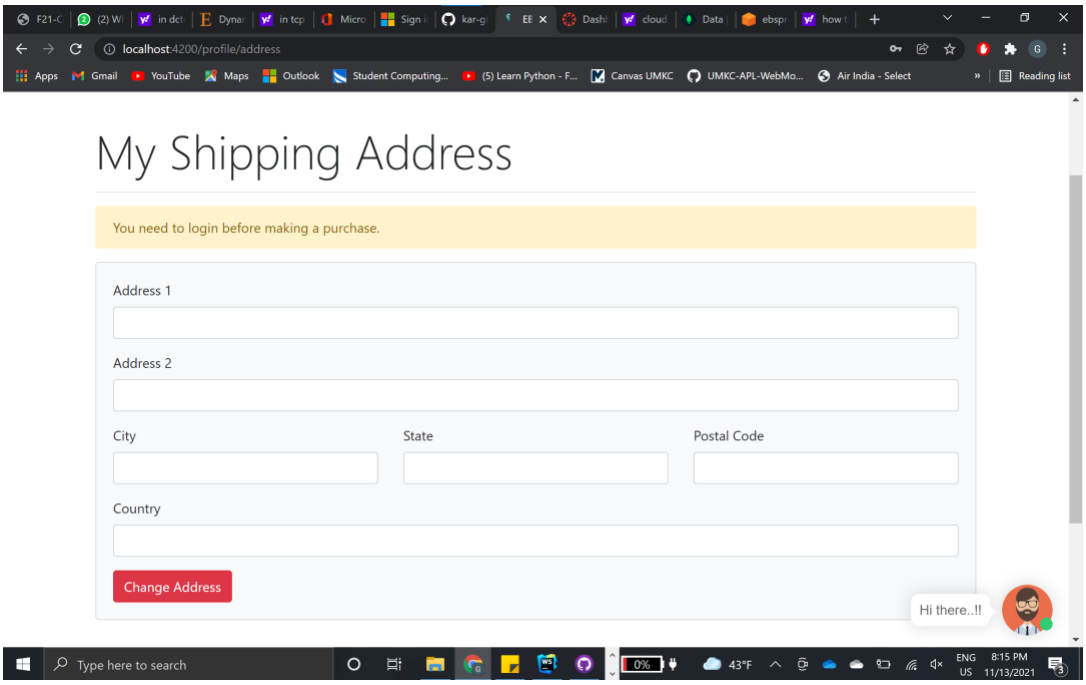
Quantity of the product can be changed as per user requirement.

Product is added to the cart.

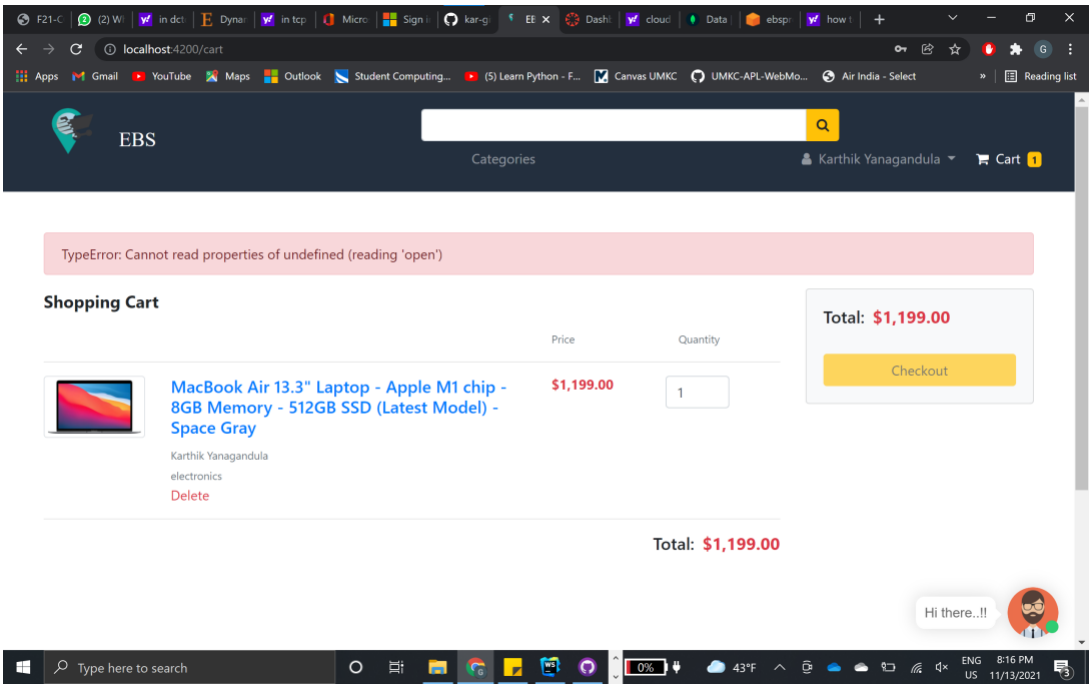


Checkout option:

When user clicks on check out, he must provide the shipping address before making a purchase.



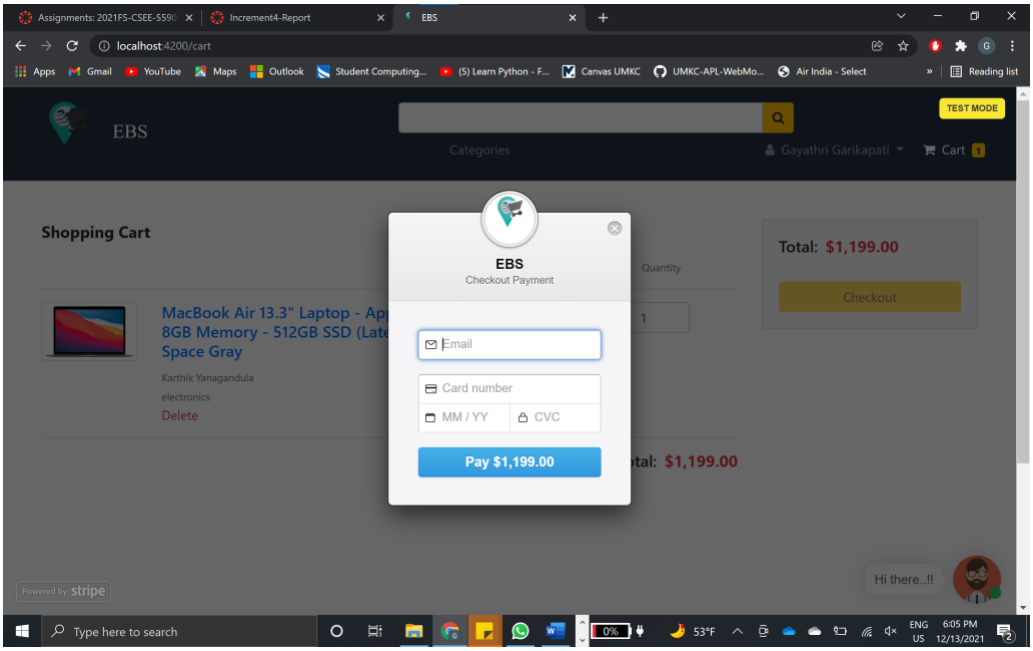
Now when we proceed to checkout for payment, we are receiving error due to Stripe API version update.

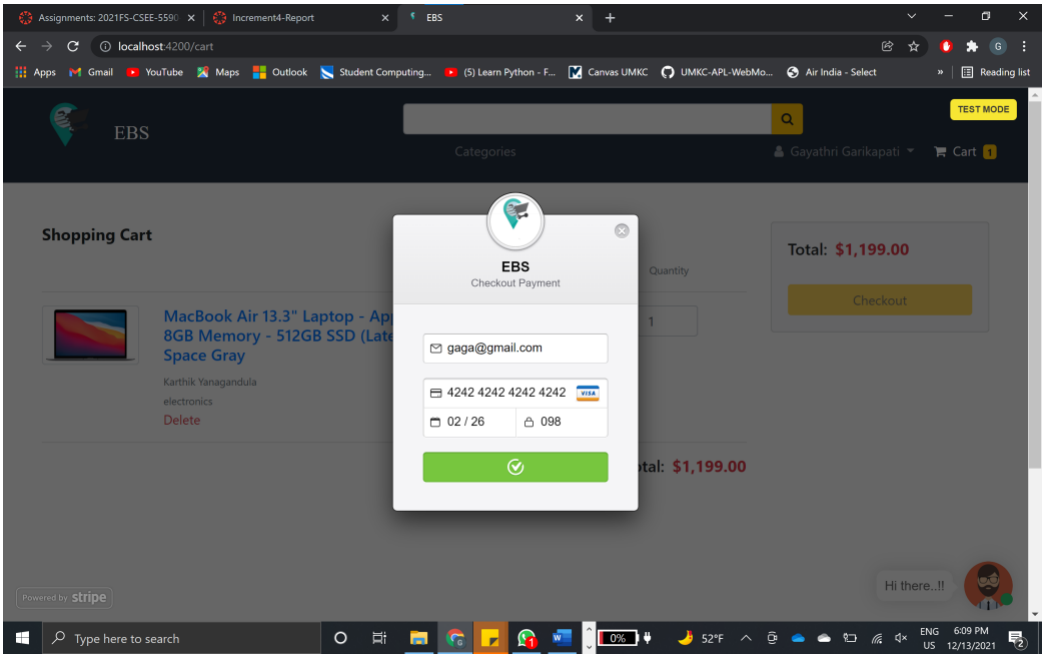


Improvement from the previous increment:

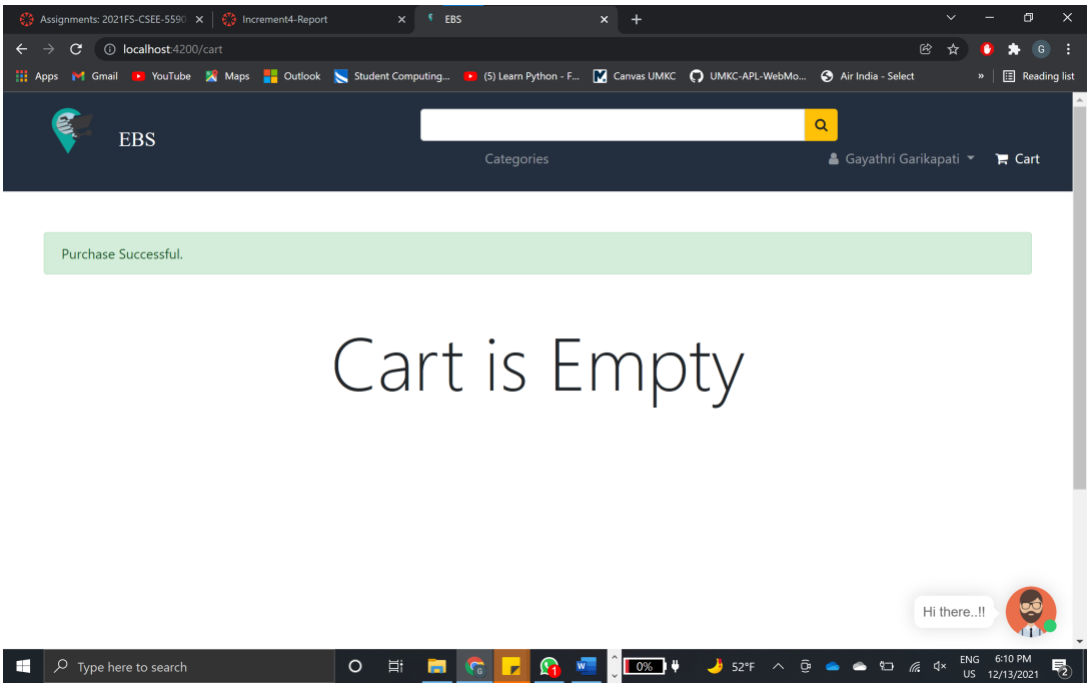
We have resolved the Stripe checkout error and were able to check out the cart and proceed with payment.

Here when the user clocks on checkout he will be redirected to the checkout payment page where the details like user's email id, card number, expiry date and security code should be provided.

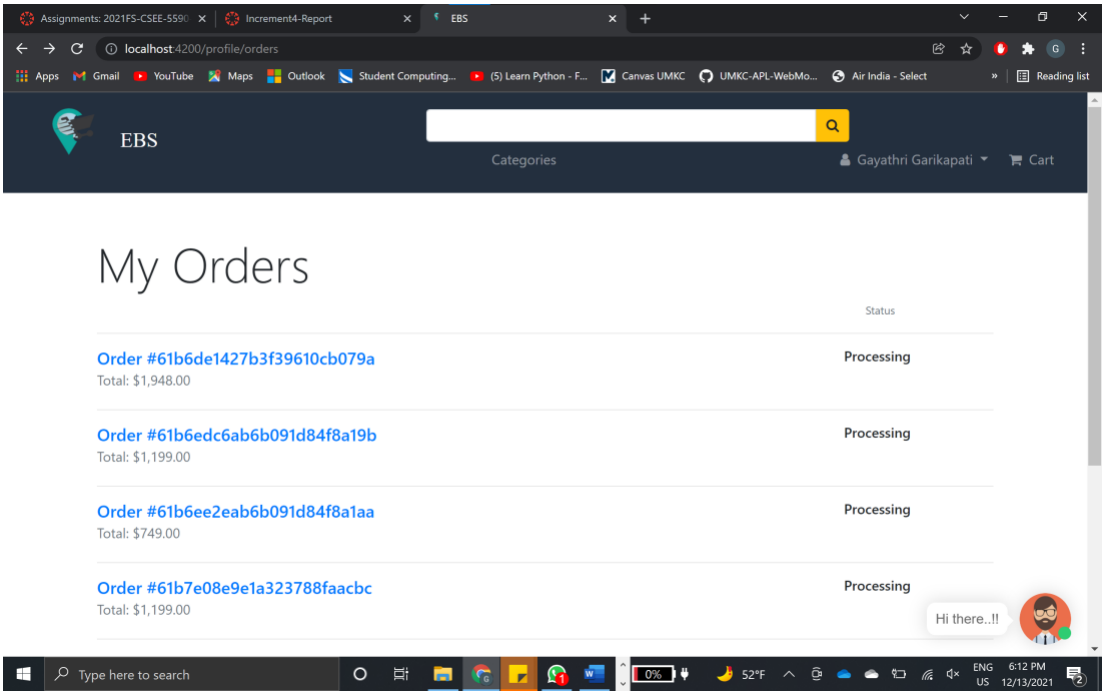


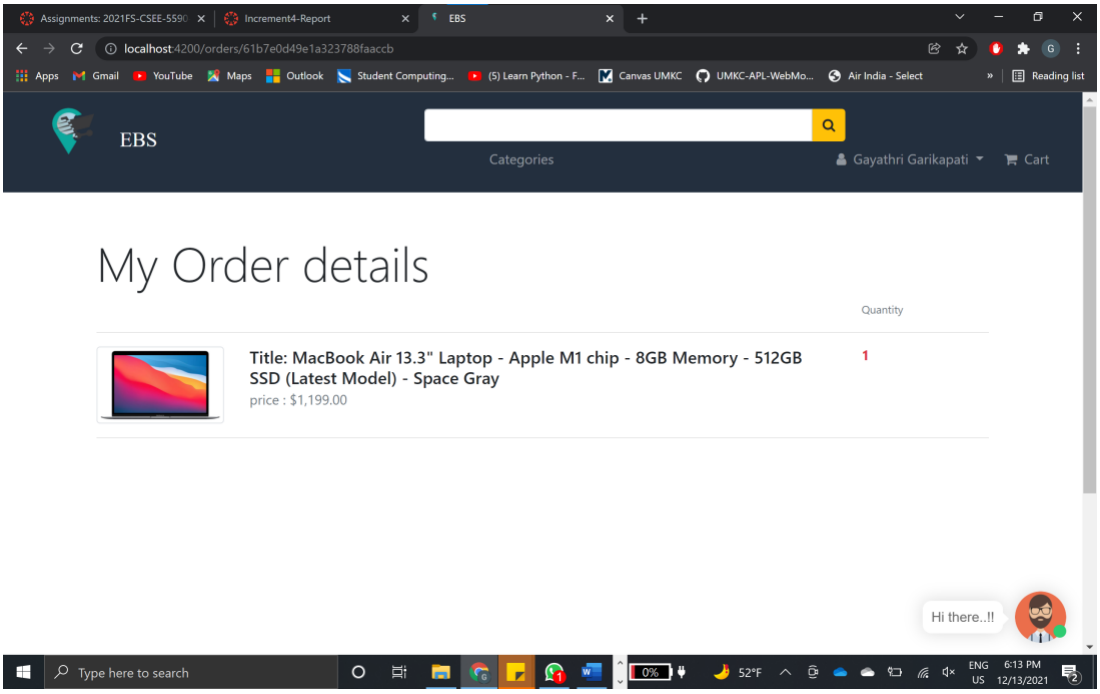


After payment the “payment successful” and the “cart is empty” message will be displayed.

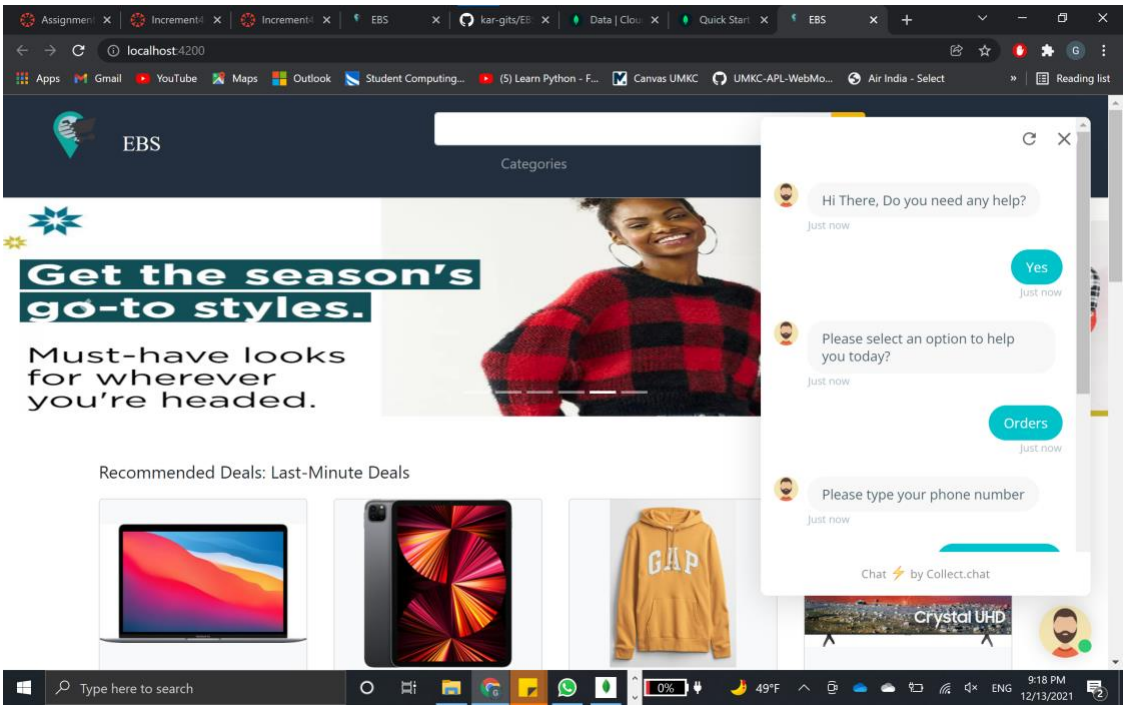


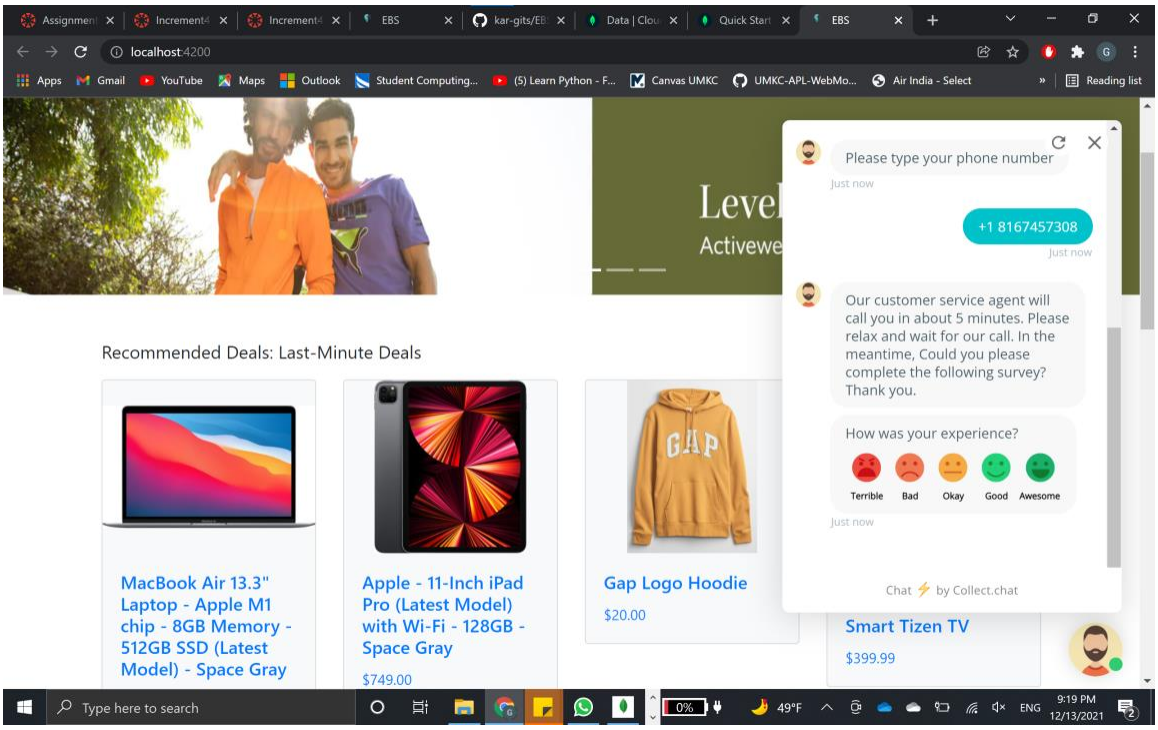
All the purchased items will be displayed under “My Orders”




EBS
Categories
Gayathri Garikapati
Cart
My Order details
Quantity
Title: MacBook Air 13.3" Laptop - Apple M1 chip - 8GB Memory - 512GB SSD (Latest Model) - Space Gray
price : $1,199.00
1
Hi there..!!
Type here to search

ChatBot:

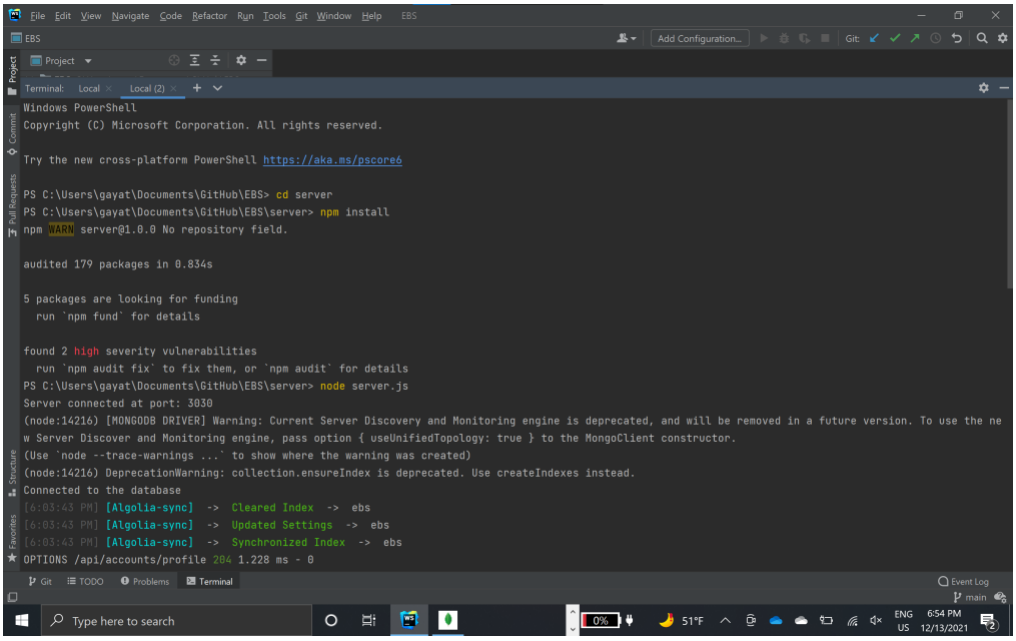

EBS
Categories
Get the season's go-to styles.
Must-have looks for wherever you're headed.
Recommended Deals: Last-Minute Deals
Hi There, Do you need any help?
Just now
Yes
Just now
Please select an option to help you today?
Just now
Orders
Just now
Please type your phone number
Just now
Chat by Collect.chat
Crystal UHD
Type here to search

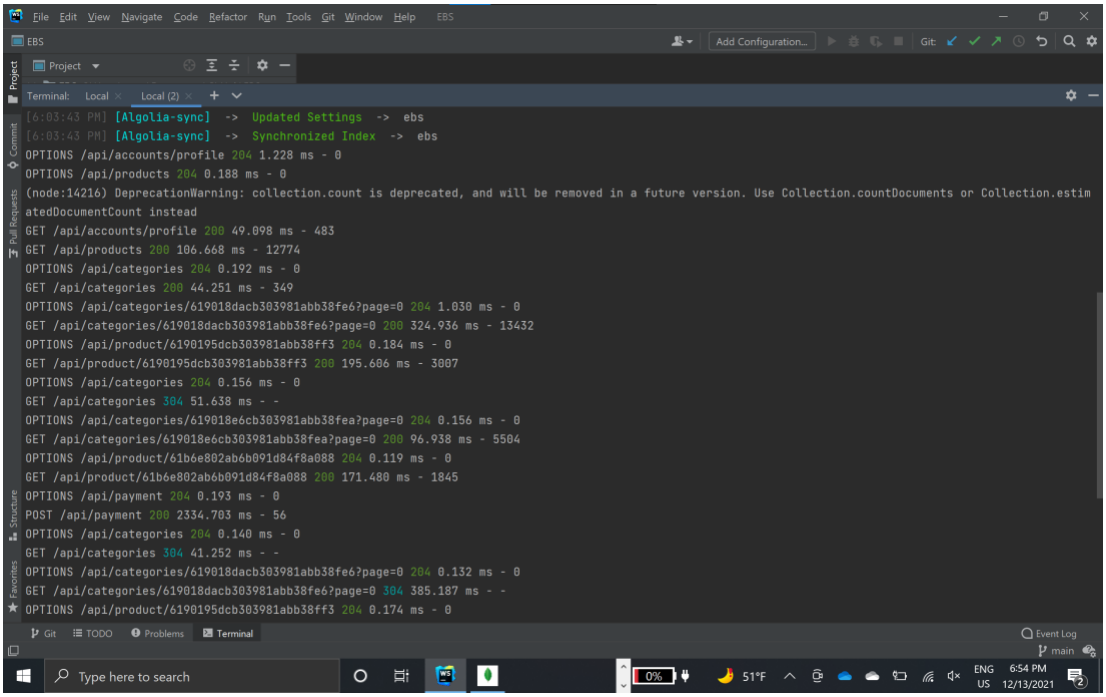


Important code snippets from the project:

EBS Server: npm install

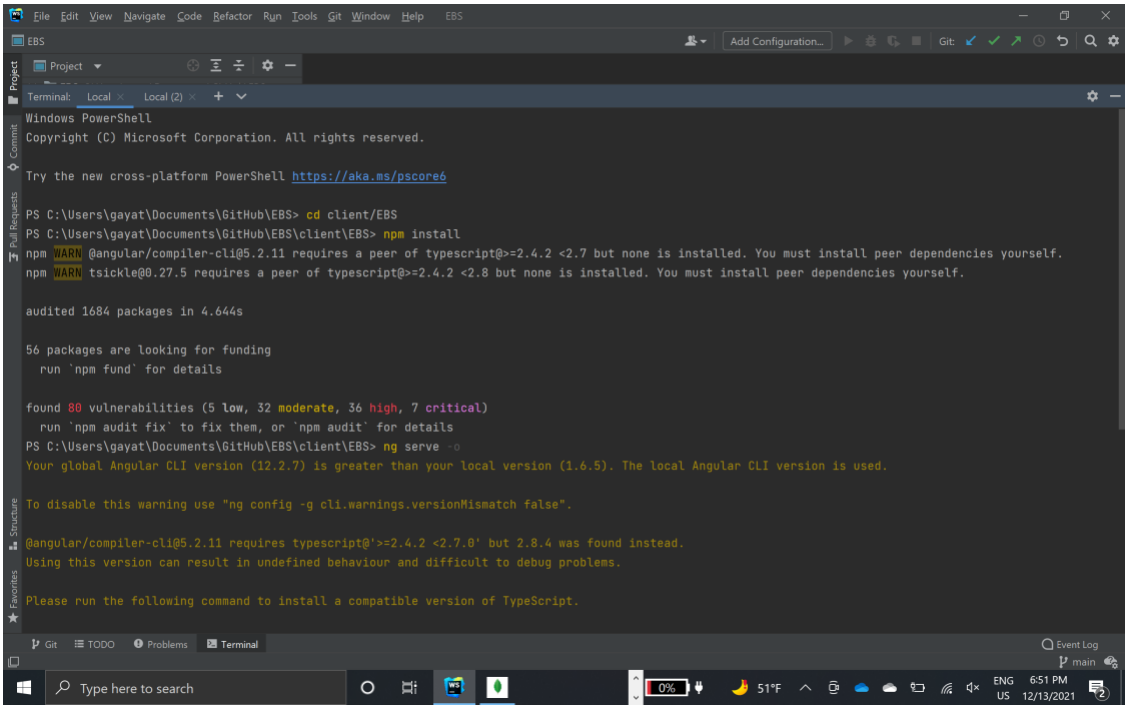
node server.js

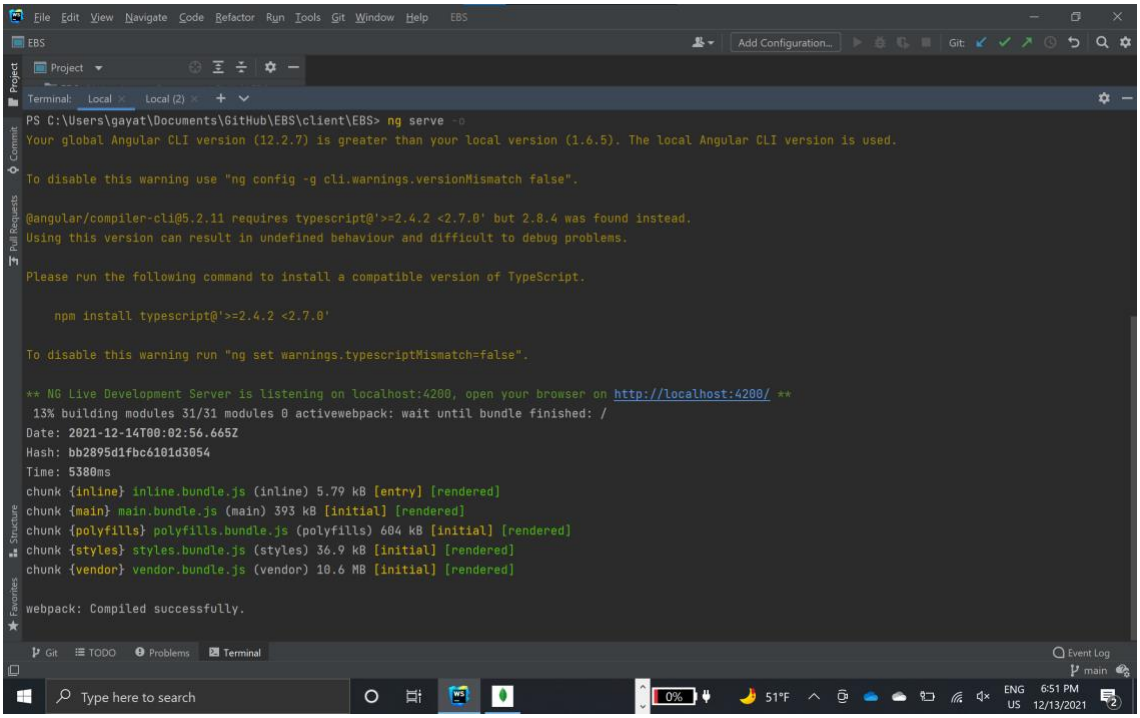




EBS client: npm install

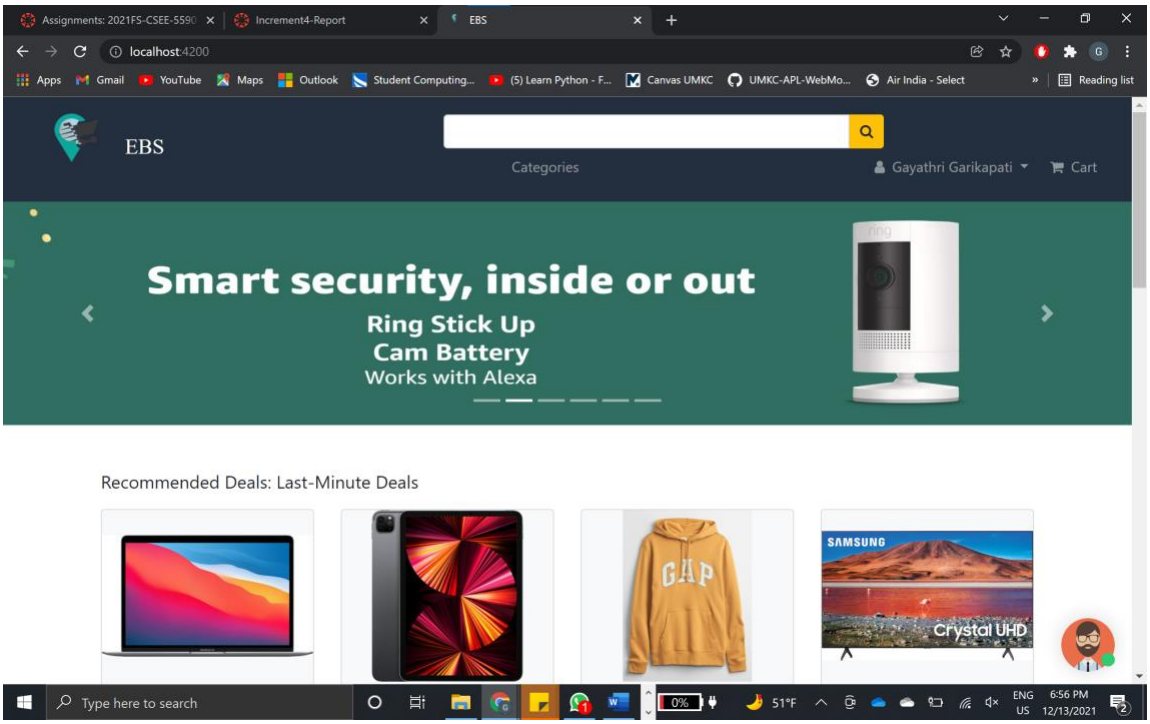
ng serve -o



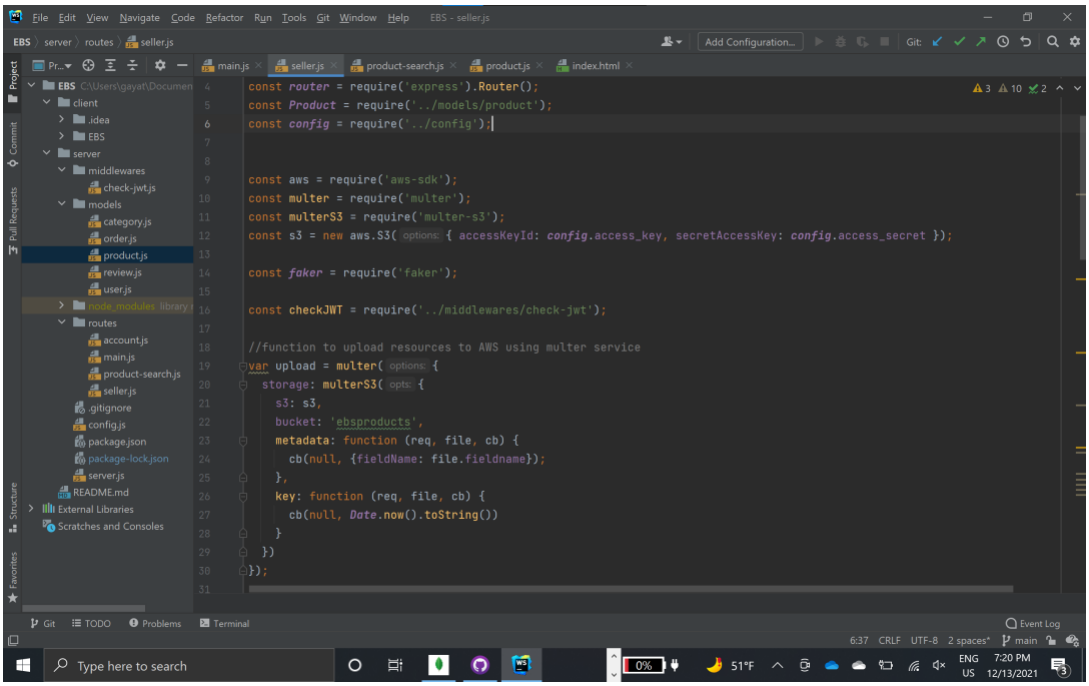


“ng serve -o” will open our application in a browser with web host

http://localhost:4200/

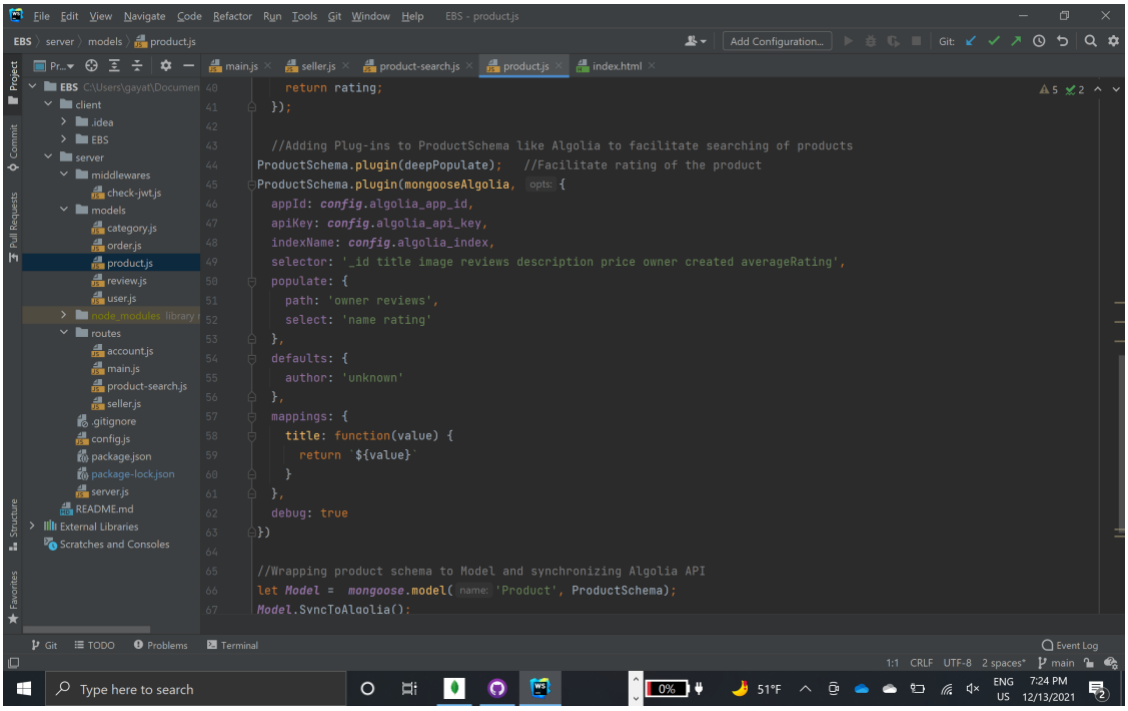


Code to maintain storing of product images on AWS. We are using mutler package to connect to AWS S3 bucket using secret keys.



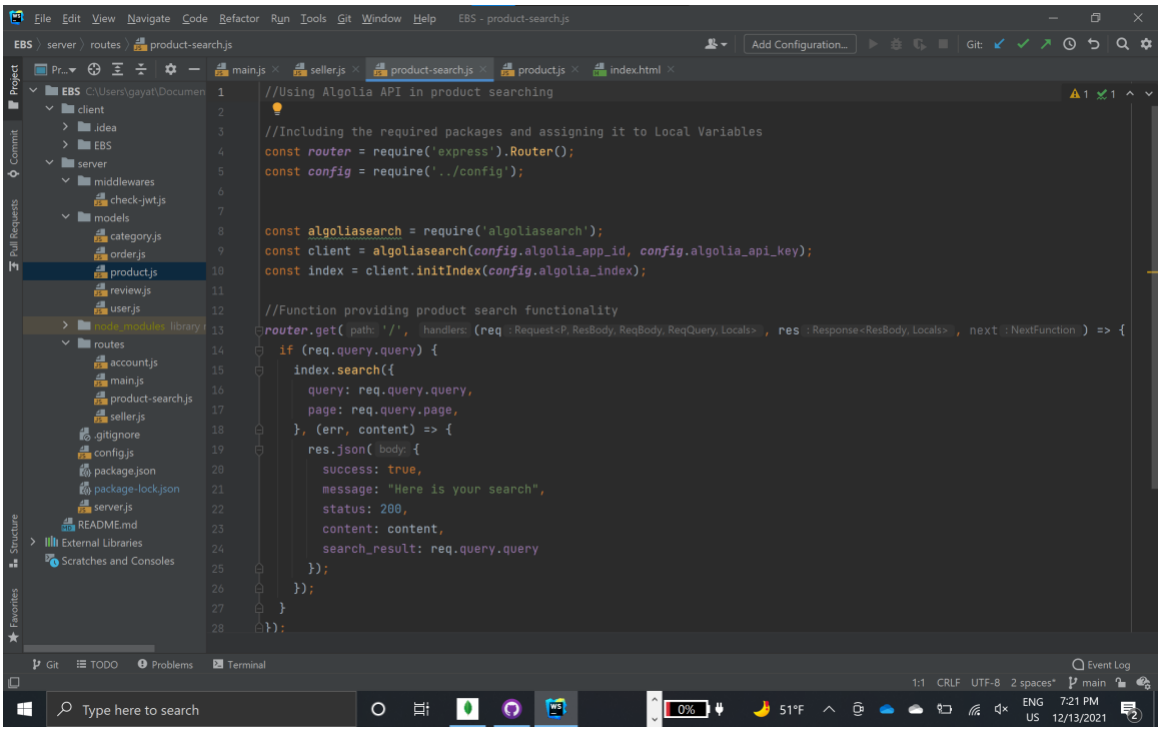
```
const router = require('express').Router();
const Product = require('../models/product');
const config = require('../config');


const aws = require('aws-sdk');
const multer = require('multer');
const multerS3 = require('multer-s3');
const s3 = new aws.S3( options: { accessKeyId: config.access_key, secretAccessKey: config.access_secret });

const faker = require('faker');

const checkJWT = require('../middlewares/check-jwt');

//function to upload resources to AWS using multer service
var upload = multer( options: {
  storage: multerS3( opts: {
    s3: s3,
    bucket: 'ebsproducts',
    metadata: function (req, file, cb) {
      cb(null, {fieldName: file.fieldname});
    },
    key: function (req, file, cb) {
      cb(null, Date.now().toString())
    }
  })
});
```

Using mongoose-algolia, we connected Algolia API to product schema, which loads the product data to Algolia search API. This ensures that all the product details in application are shared with API to perform search operations.



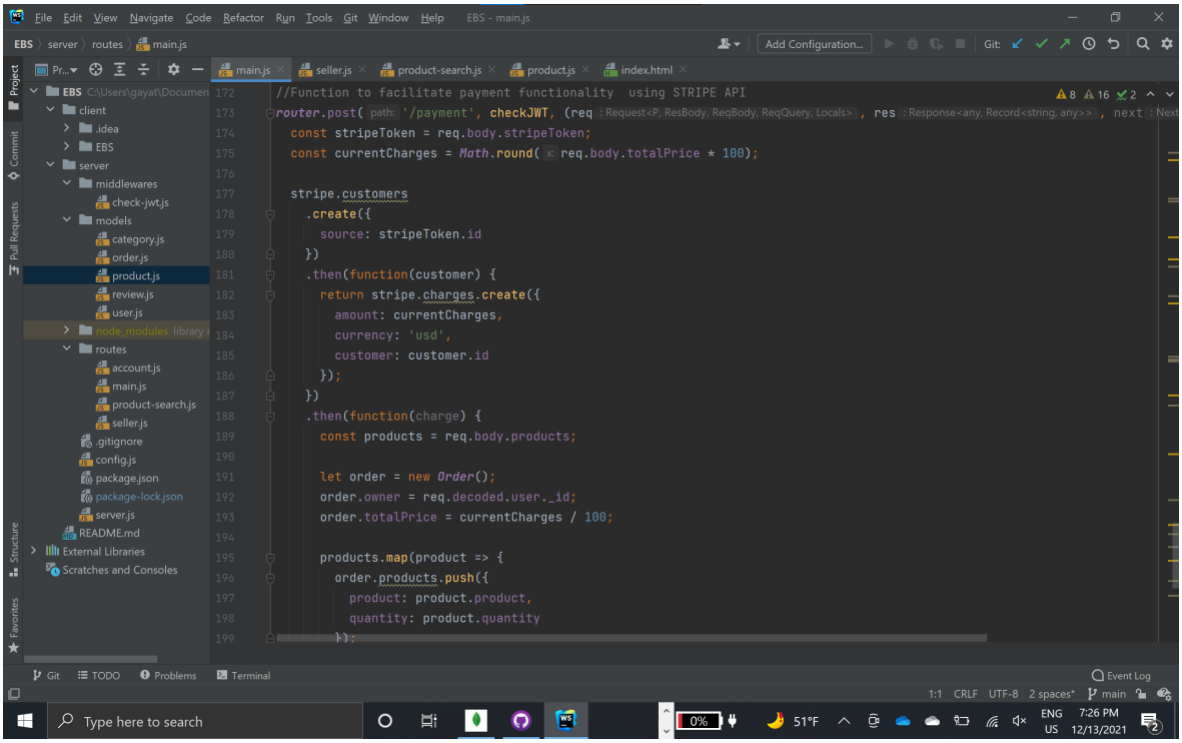
```
    return rating;
  });

  //Adding Plug-ins to ProductSchema like Algolia to facilitate searching of products
ProductSchema.plugin(deepPopulate);   //Facilitate rating of the product
ProductSchema.plugin(mongooseAlgolia,  opts: {
  appId: config.algolia_app_id,
  apiKey: config.algolia_api_key,
  indexName: config.algolia_index,
  selector: '_id title image reviews description price owner created averageRating',
  populate: {
    path: 'owner reviews',
    select: 'name rating'
  },
  defaults: {
    author: 'unknown'
  },
  mappings: {
    title: function(value) {
      return `${value}`
    }
  },
  debug: true
})

//Wrapping product schema to Model and synchronizing Algolia API
let Model =  mongoose.model( name: 'Product', ProductSchema);
Model.SyncToAlgolia();
```

To facilitate product search using Algolia search API. We are using secret keys and index name of search API to perform the task.

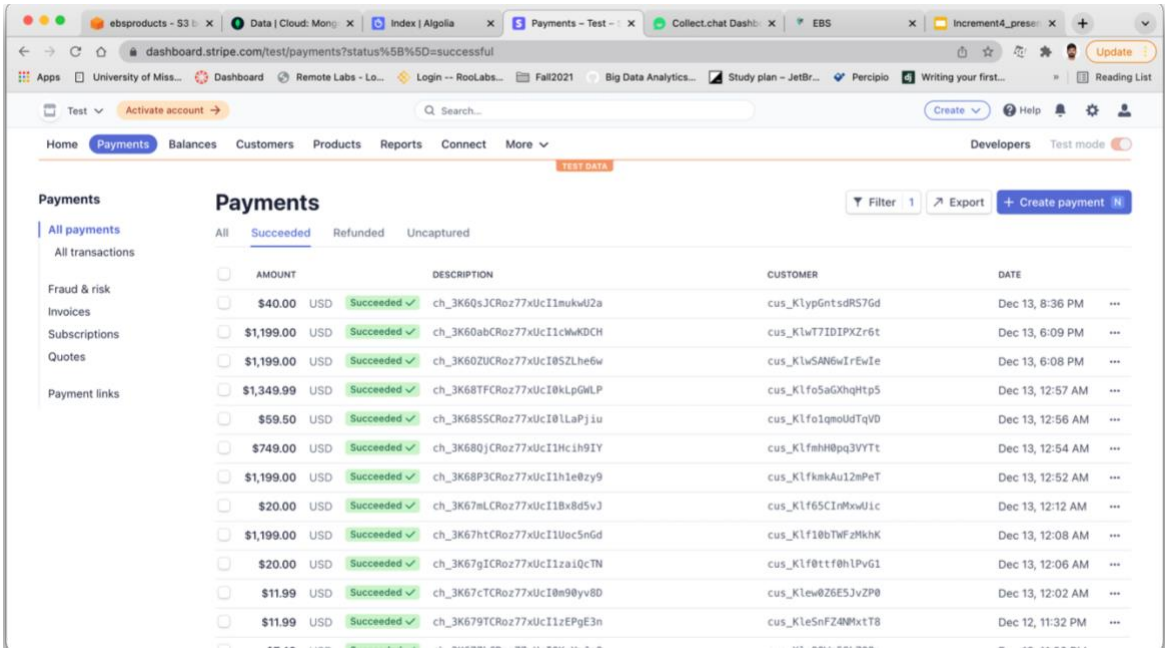


```
//Using Algolia API in product searching

//Including the required packages and assigning it to Local Variables
const router = require('express').Router();
const config = require('../config');


const algoliasearch = require('algoliasearch');
const client = algoliasearch(config.algolia_app_id, config.algolia_api_key);
const index = client.initIndex(config.algolia_index);

//Function providing product search functionality
router.get( path: '/', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> , next : NextFunction ) => {
  if (req.query.query) {
    index.search({
      query: req.query.query,
      page: req.query.page,
    }, (err, content) => {
      res.json( body: {
        success: true,
        message: "Here is your search",
        status: 200,
        content: content,
        search_result: req.query.query
      });
    });
  }
});
```

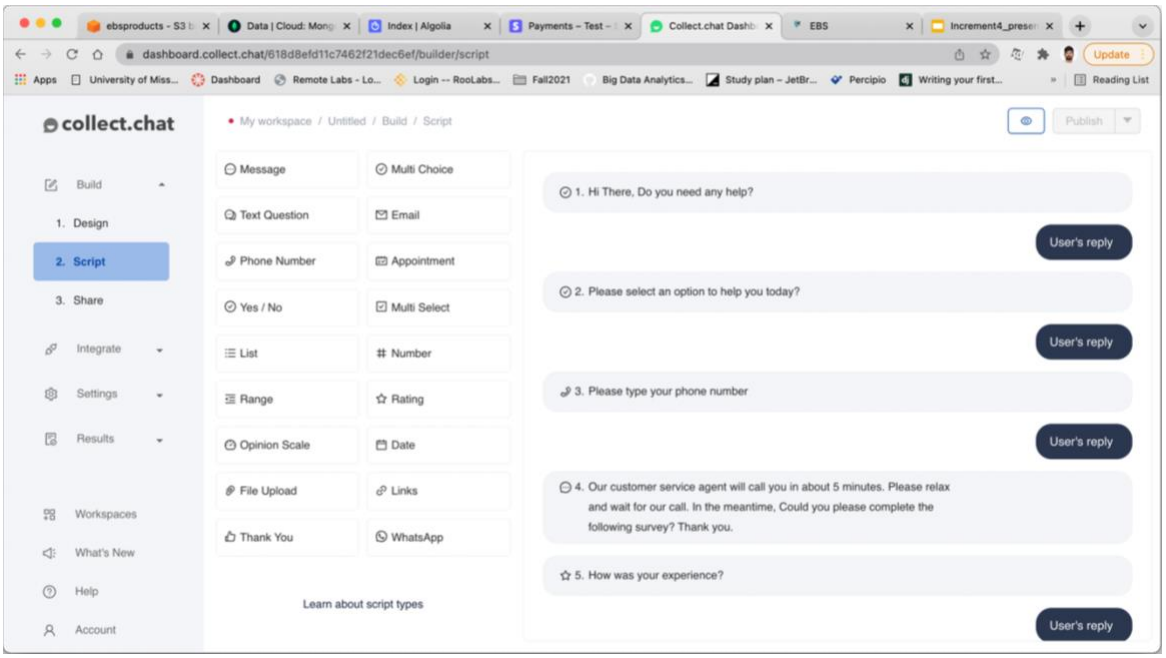
Code to facilitate payment functionality using STRIPE API

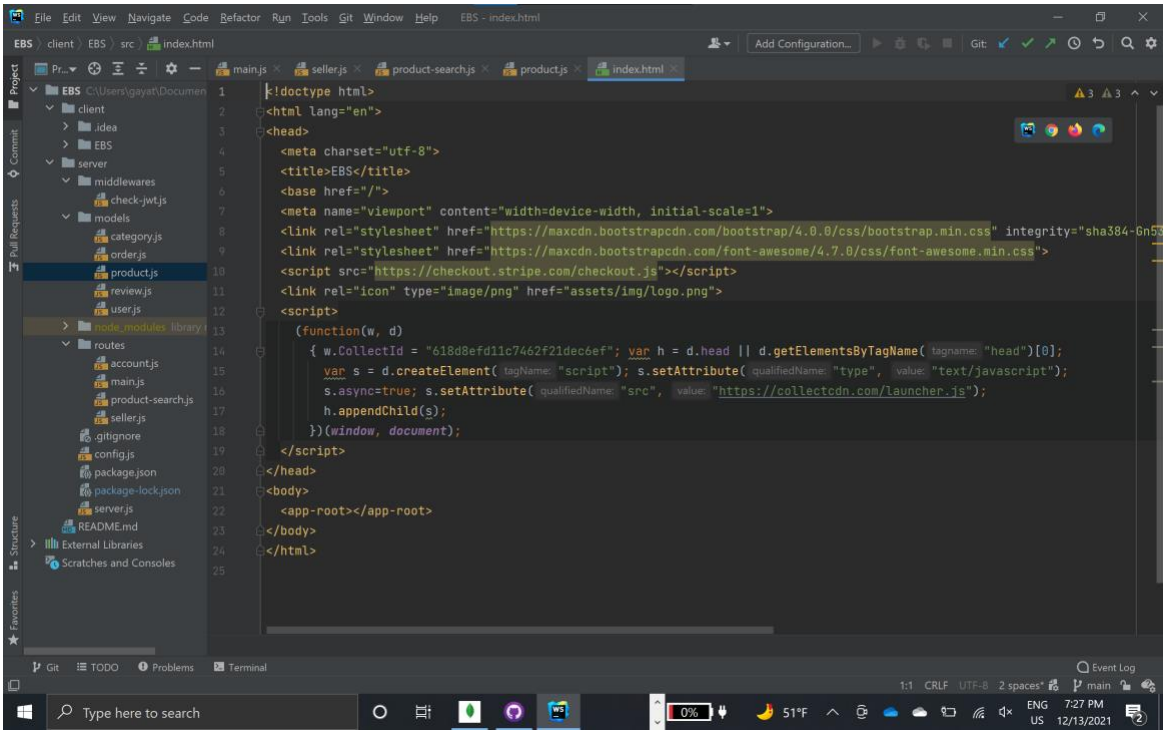


```
//Function to facilitate payment functionality  using STRIPE API
router.post( path: '/payment', checkJWT, (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<any, Record<string, any>> , next : Next
  const stripeToken = req.body.stripeToken;
  const currentCharges = Math.round( x: req.body.totalPrice * 100);

  stripe.customers
    .create({
      source: stripeToken.id
    })
    .then(function(customer) {
      return stripe.charges.create({
        amount: currentCharges,
        currency: 'usd',
        customer: customer.id
      });
    })
    .then(function(charge) {
      const products = req.body.products;

      let order = new Order();
      order.owner = req.decoded.user._id;
      order.totalPrice = currentCharges / 100;

      products.map(product => {
        order.products.push({
          product: product.product,
          quantity: product.quantity
        });
```



To implement ChatBot functionality, we included the script generated in collect.chat dashboard.





Work Sharing:

Vidya D: Worked on profile and review components, tried to implement multiple shipping address feature for placing order.

Karthik Y: Worked on database models, storing database on cloud and backend code for routes. Worked on Stripe Payment API issue and added simple chatbot using collect.chat website.

Gayathri G: Worked on adding Algolia search API for product search, account creation, logging in and product feature implementation.

Pujitha T: Worked on post products, account settings and order details. Completed checking the orders in database and UI after Stripe issue is resolved.

Common Work: Worked on report and presentation for the work said above.

Issues with the project:

Due to a version change for the Stripe API, we had an issue processing payment due to a "Stripcheckout" error during the product checkout. We were able to find a solution to the problem. We didn't have enough time to add the additional capabilities to the application because the end-to-end work took so long, therefore we'll try to incorporate these functionalities (Split payments and multiple address delivery) as part of future work.

Future work:

- Split payment
- Product delivery to multiple address with single checkout

GitHub link:

https://github.com/kar-gits/EBS

Video:

https://youtu.be/bobL7H5Y2ac

Presentation:

https://docs.google.com/presentation/d/1xG7kWOmHmBk-elTSNinDcacMdJQOx1r1/edit?usp=sharing&ouid=107184809928603804089&rtpof=true&sd=true

References:

- https://www.algolia.com/doc/rest-api/search/
- https://stripe.com/docs/api
- https://angular.io/docs
- https://www.mongodb.com/developer/learn/?content=Articles&text=Node.js
- https://nodejs.org/dist/latest-v14.x/docs/api/synopsis.html
- https://collect.chat/templates/