Introduction :

E-commerce is fast gaining ground as an accepted and used business paradigm. More and more business houses are implementing web sites providing functionality for performing commercial transactions over the web. It is reasonable to say that the process of shopping on the web is becoming commonplace.

This project is centered on e-commerce, and Here we intend to create a website for an online grocery store. Thanks to the Internet, all of our lives are easier now. Nowadays, practically anything can be done online, including grocery shopping. Today, many people choose to shop for their groceries online. With the use of this web application, customers can purchase daily necessities like groceries online. This comprises bread, fruits, vegetables, and other foods. It is very important to store the data in a specific format that can be accessible and manageable by our web application.

Every organization, whether big or small, has challenges to overcome and manage the information of customers, Grocery, stock, Address, and products. We believe choosing The database is the best option to do so. This web application will be using the NoSQL database, by which we can store the necessary data and perform Create, Read, Delete, and Update operations quickly and efficiently. Unlike traditional relational databases. NoSQL (MongoDB) provides high scalability and availability.

Objective of project :

The objective of this project is to develop a general-purpose e-commerce store where any product (such as books, CDs, computers, mobile phones, electronic items, and home appliances) can be bought from the comfort of home through the Internet. However, for implementation purposes, this project will deal with some extent of products from various categories.

We are trying to implement a No-SQL database for an e-commerce site. It is nothing new but I believe it is a good way to start learning non-relational Databases. Since we can go through the functionalities of many web applications like amazon, eBay, Walmart, etc., This Application should be a mini replica of many of the applications online with some standards. Our web application will be using collections like “products”, ” users”, ” payments”, ”orders” and “Order status”.

Database Description and Business Rules:

This is an E-commerce web application. We are using No-sql database model by taking references from several applications like amazon, flipkart, walmart etc. Using this application people can sign-up and login as a customer to keep track of every individual details and previous activity. We can populate all the products using the products collection but Only registered users can place the orders. We will be maintaining an inventory of all the products with several attributes to keep track of the products. Each product will have a unique item_id, category, Available quantity etc. we have two types of customers , general users and people with admin access these people will have admin permissions using which they can perform crud operation on products and will have restricted permission on all the other collections.

Users can login to their accounts using their respective username and password. After sign-in they can view the products, add the products to the cart, and also remove the products from the cart. There is no need to have a separate collection for this we will be using session and cookies functionality to store the items in the local browser storage.Each customer can have any number of orders and each order can have any number of products.We can track each order using a unique order_id and can know the order status. After adding the products to the cart in an order users can checkout by proceeding to the payment or can delete the order itself. Then we need to calculate the total price of the order based on individual items and the quantity which will be obtained using order_id. We will get the payment information like payment_method, type of card and shipping information from the customer collection. Then we go through the payment process. We can know the status of the payment using the payment_status attribute in the payment collection. Users can cancel the order after 24 hours of the payment. After the delivery of the order, customers can return products which come under specific constraints like users cannot return products which come under the category of vegetables,meat etc.

Users can return some products (cosmetics) before a specific span of time from the day of delivery. Customers can know the return status with the help of return_status attribute in returns collection.

Technologies Used : Python, MongoDB , Flask , Html , CSS Bootstrap.

Collections :

- **Orders** collection contains Order_id, Email_id,total_price, time_order, Order_status. Item_details[] array which contains a list of item_id, Qty, discount,price and category.
- **Customers** collection contains Email_id(userName), Password, First_name, Last_name, PhoneNo, Address array including country, street1, street2, city, state, zip and Shipping_Address [] Array which contains same attributes as aAddress Array and Payment_details[] Array which contains cards details of the customer namely Name ,card_type and exp_date.
- **Products** collection contains item_id, Title, description, price,quantity, category, discount and sku{}.
- **Payment** collection contains Payment_id, order_id,email_id, payment_details[] Array which includes card_Type,exp_date and name, payment_status,time_payment,total_price_tax and ShippingAddress array including country, street1, street2, city, state, zip.
- **Returns** collections contain Return_id, order_id, payment_id,email_id,return_status and return_payment_method.
- **Admin** collection contains name, username(email_id), password and phone number.

Functionalities of Admin :

- The Admin will login to the system using their respective Email_Id and Password.
- Admin can Add inventory (products) to the Database for the ecommerce

site.

- Admin can View all the inventory of the ecommerce site and also can update the existing products details like product_title, Description, Quantity and price etc.,
- Admin can view all the orders.
- Admin can also view the return product requested by the customers and Accept or Reject the return requests.

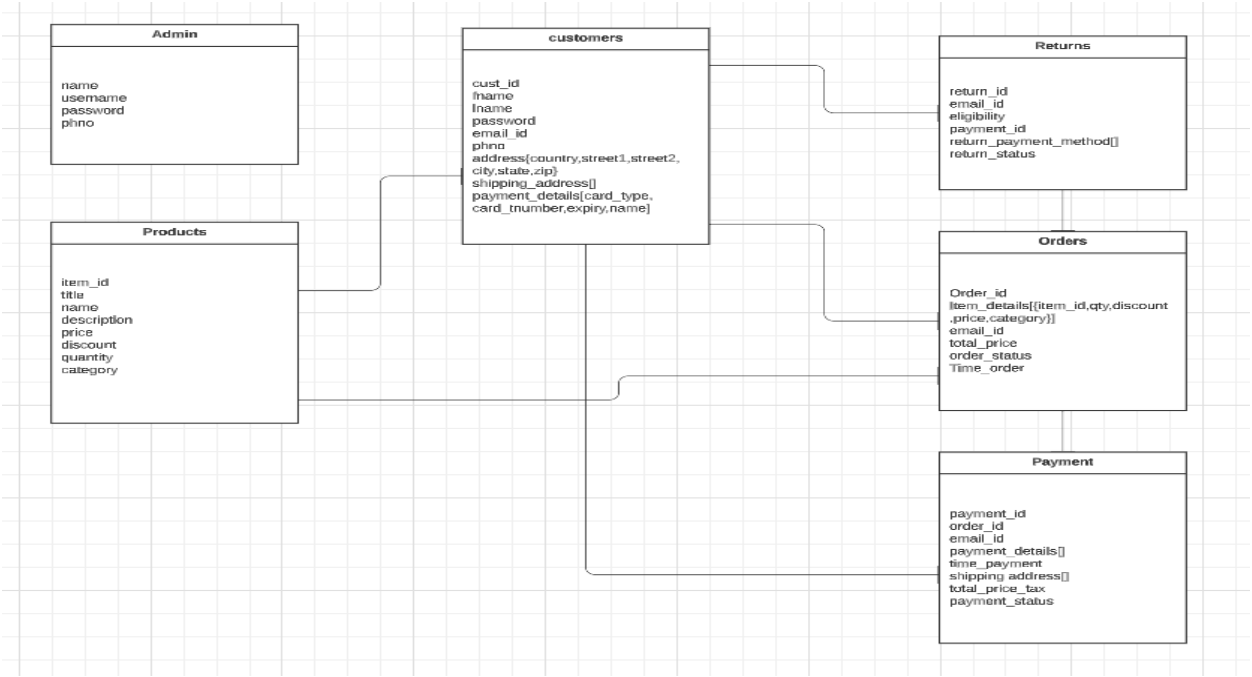
Functionalities of Customer :

- Customers need to be Registered to order the products by filling the sign-up form.
- Only Registered Customers can Login to the system using Email_id and password to access their respective Accounts.
- Customers can view all the products present in the home page.
- They can choose the products with quantity which are added to the cart.
- They can proceed to pay and order the products in the cart or remove some products and proceed.
- After payment is successful the product's quantity details are updated in the inventory.
- If the product is out of stock, customers cannot add those products to the cart.
- Customers can view their orders from my_orders page.
- Customers can also raise a return request for a delivered product, the decision to accept or reject the return is handled by the admin.
- If the return is accepted then the product quantity is added into the inventory otherwise no change.
- Customers will have restrictions on return products like tim_based and

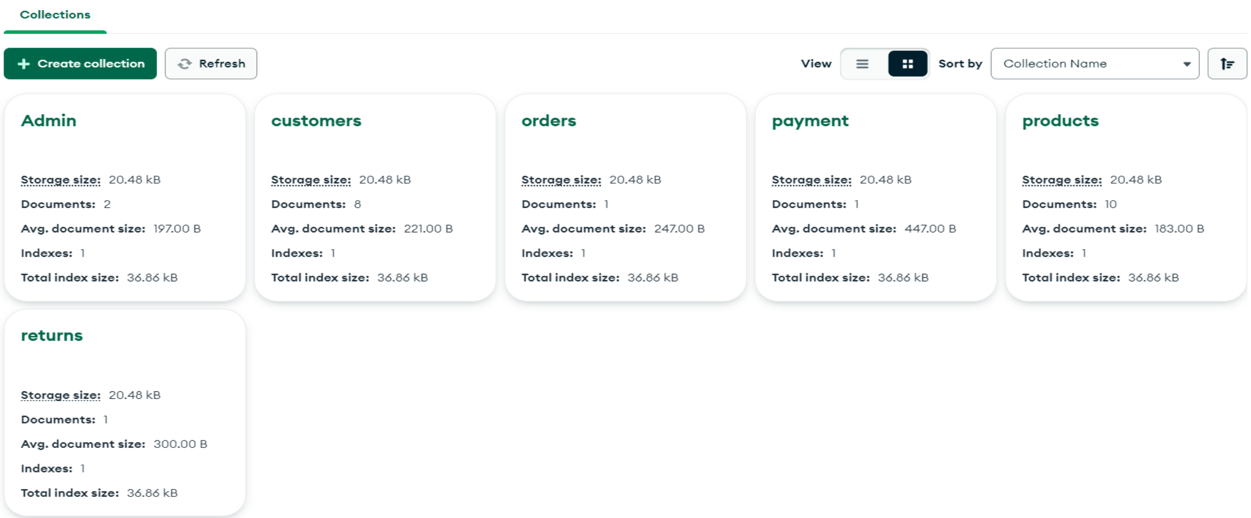
category base etc.,

- Customers can check their returns status from my_returns page.

Database Model :



Collections sample page :



Admin

Storage size: 20.48 kB
Documents: 2
Avg. document size: 197.00 B
Indexes: 1
Total index size: 36.86 kB

customers

Storage size: 20.48 kB
Documents: 8
Avg. document size: 221.00 B
Indexes: 1
Total index size: 36.86 kB

orders

Storage size: 20.48 kB
Documents: 1
Avg. document size: 247.00 B
Indexes: 1
Total index size: 36.86 kB

payment

Storage size: 20.48 kB
Documents: 1
Avg. document size: 447.00 B
Indexes: 1
Total index size: 36.86 kB

products

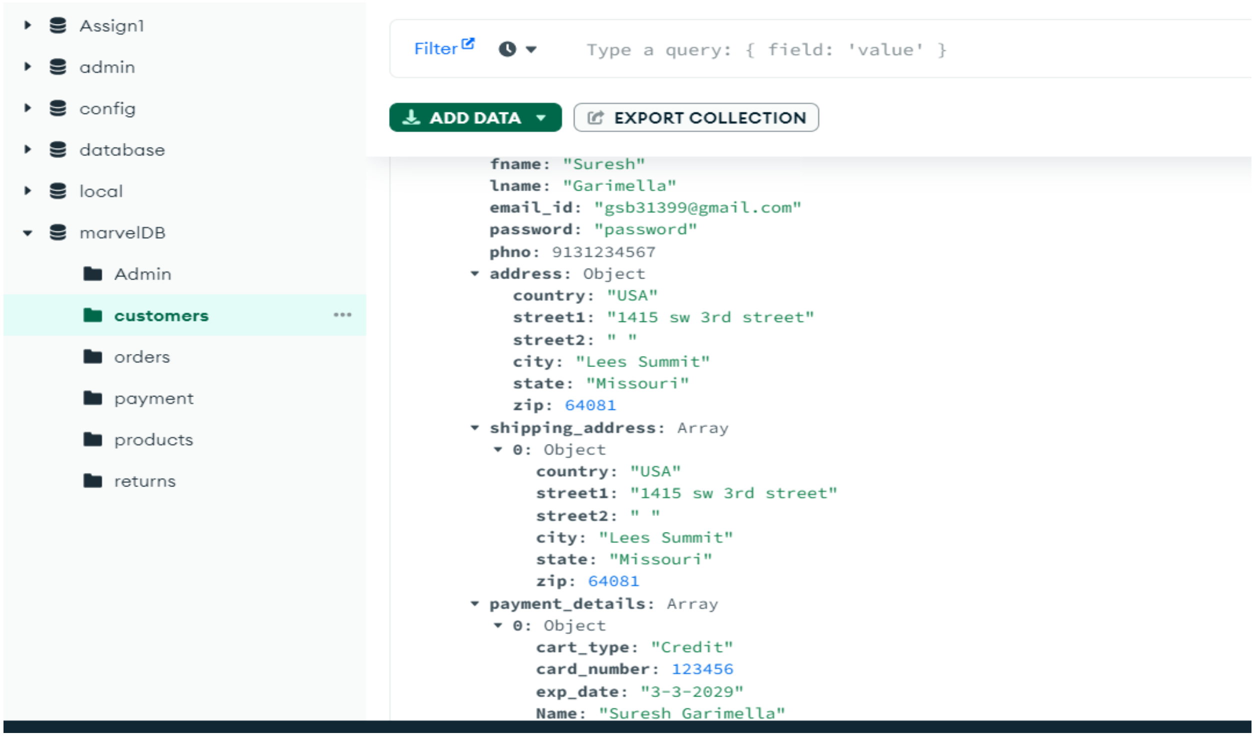
Storage size: 20.48 kB
Documents: 10
Avg. document size: 183.00 B
Indexes: 1
Total index size: 36.86 kB

returns

Storage size: 20.48 kB
Documents: 1
Avg. document size: 300.00 B
Indexes: 1
Total index size: 36.86 kB

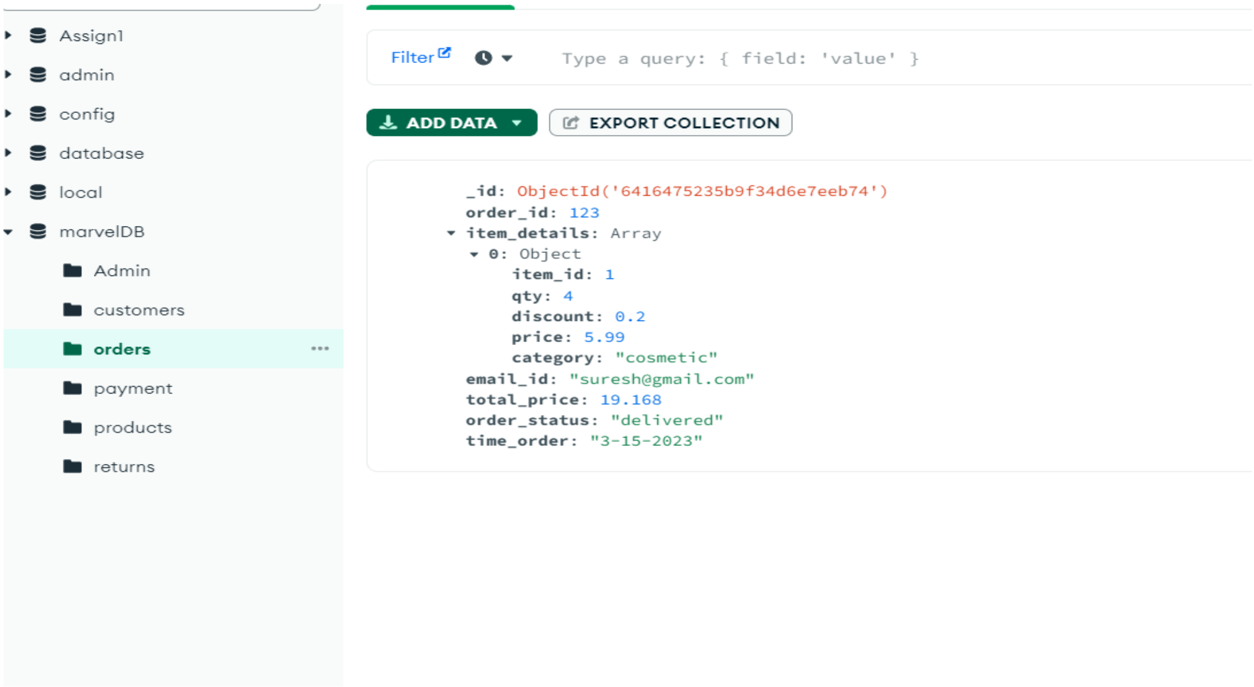
Sample data :

Customers-



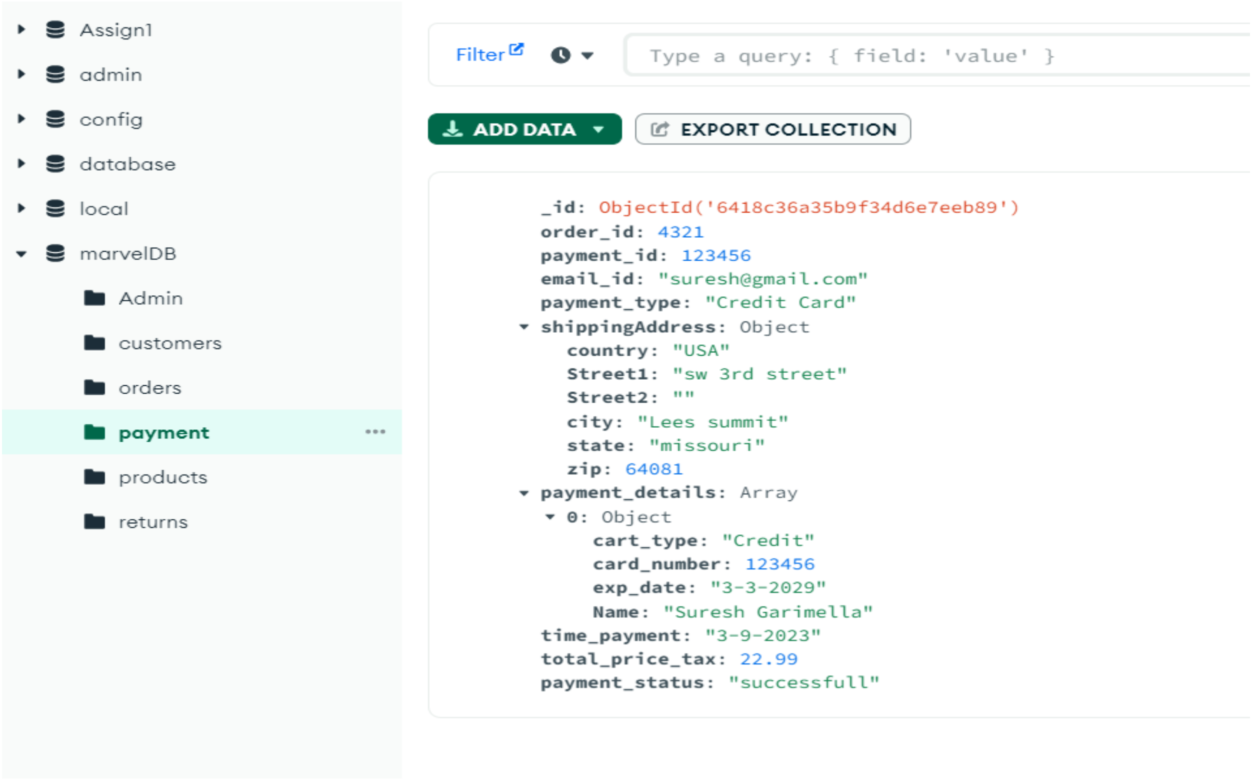
```
fname: "Suresh"
lname: "Garimella"
email_id: "gsb31399@gmail.com"
password: "password"
phno: 9131234567
▾ address: Object
    country: "USA"
    street1: "1415 sw 3rd street"
    street2: " "
    city: "Lees Summit"
    state: "Missouri"
    zip: 64081
▾ shipping_address: Array
  ▾ 0: Object
      country: "USA"
      street1: "1415 sw 3rd street"
      street2: " "
      city: "Lees Summit"
      state: "Missouri"
      zip: 64081
▾ payment_details: Array
  ▾ 0: Object
      cart_type: "Credit"
      card_number: 123456
      exp_date: "3-3-2029"
      Name: "Suresh Garimella"
```

Orders-



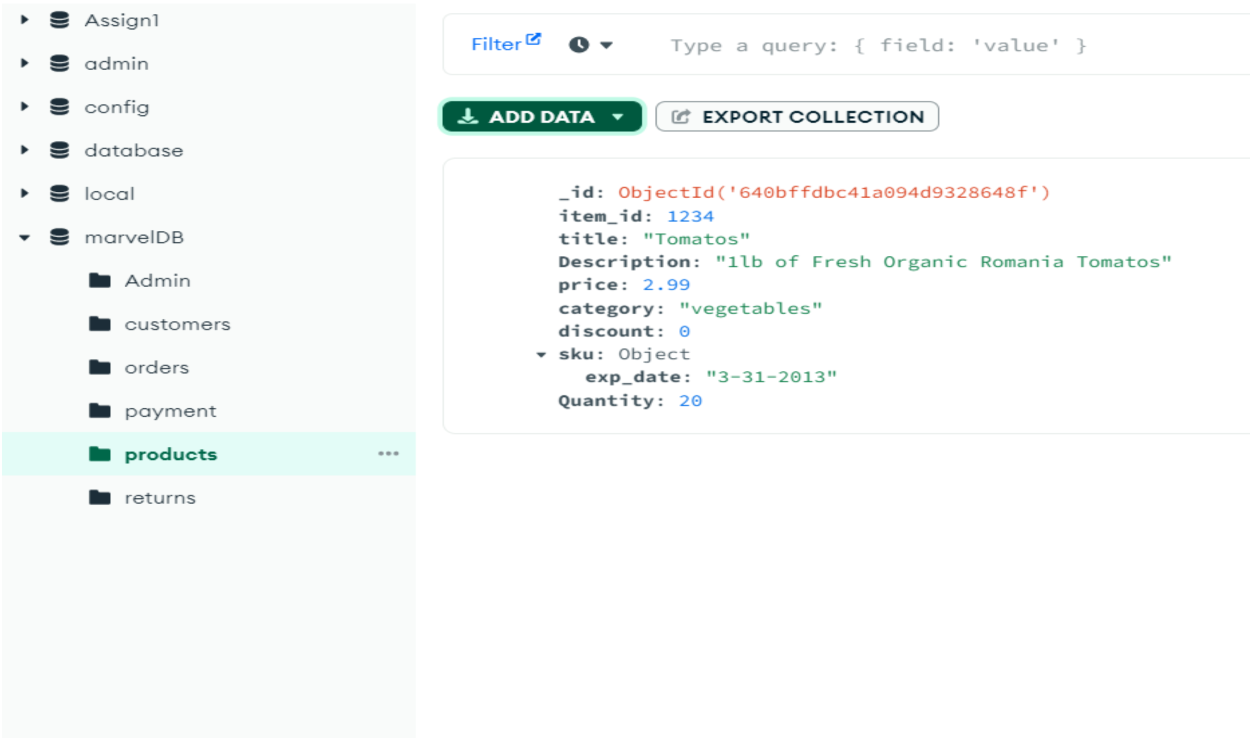
```
_id: ObjectId('6416475235b9f34d6e7eeb74')
order_id: 123
item_details: Array
  0: Object
    item_id: 1
    qty: 4
    discount: 0.2
    price: 5.99
    category: "cosmetic"
email_id: "suresh@gmail.com"
total_price: 19.168
order_status: "delivered"
time_order: "3-15-2023"
```

Payment-



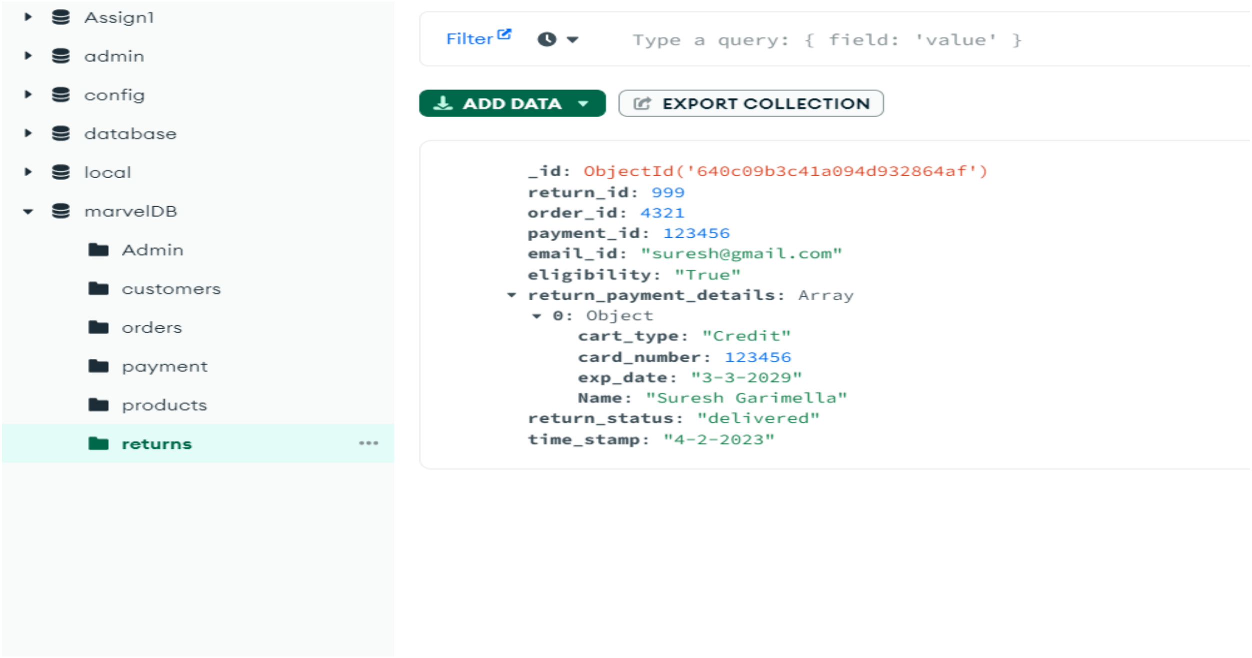
```
_id: ObjectId('6418c36a35b9f34d6e7eeb89')
order_id: 4321
payment_id: 123456
email_id: "suresh@gmail.com"
payment_type: "Credit Card"
shippingAddress: Object
  country: "USA"
  Street1: "sw 3rd street"
  Street2: ""
  city: "Lees summit"
  state: "missouri"
  zip: 64081
payment_details: Array
  0: Object
    cart_type: "Credit"
    card_number: 123456
    exp_date: "3-3-2029"
    Name: "Suresh Garimella"
time_payment: "3-9-2023"
total_price_tax: 22.99
payment_status: "successfull"
```

Products -



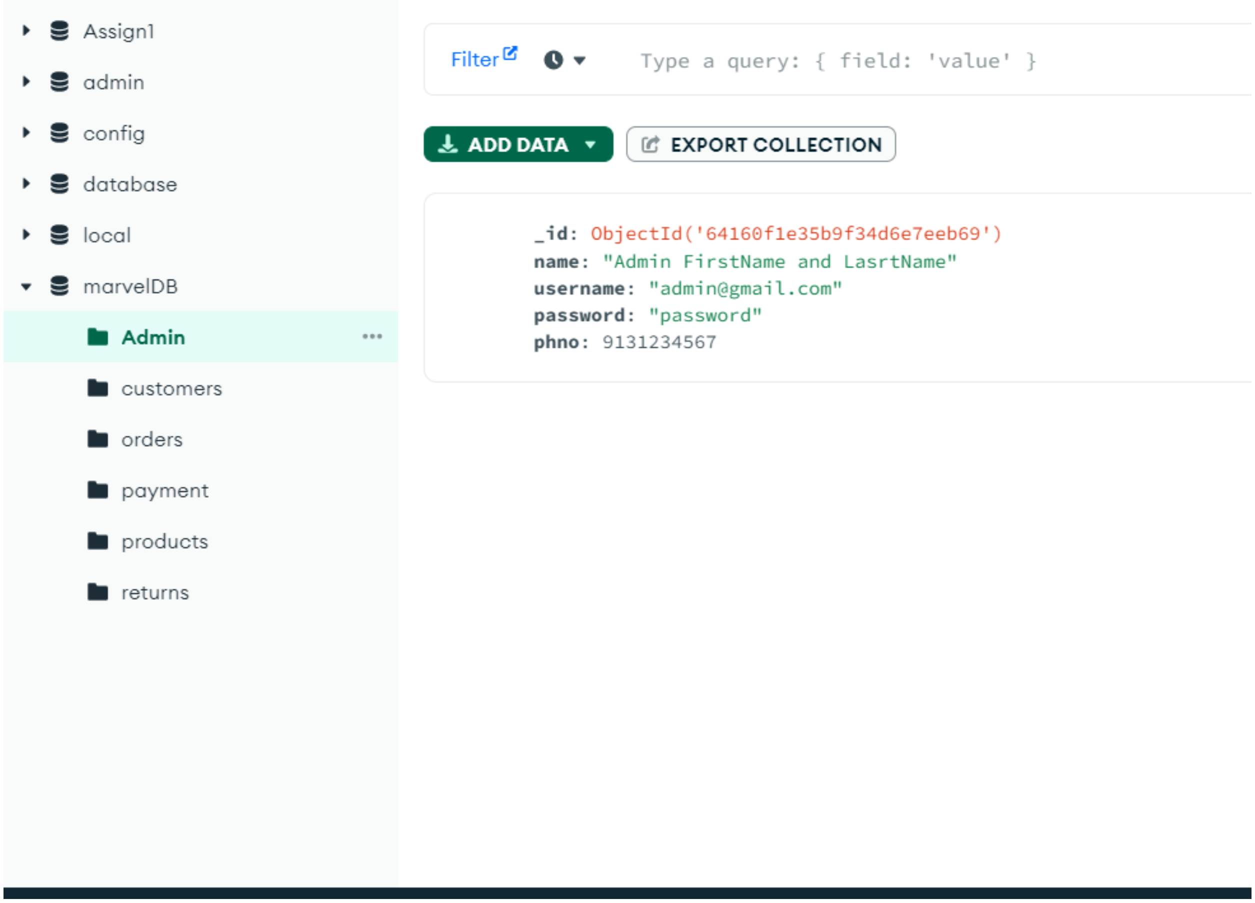
```
_id: ObjectId('640bffdbc41a094d9328648f')
item_id: 1234
title: "Tomatos"
Description: "1lb of Fresh Organic Romania Tomatos"
price: 2.99
category: "vegetables"
discount: 0
sku: Object
  exp_date: "3-31-2013"
Quantity: 20
```

Returns-



```
_id: ObjectId('640c09b3c41a094d932864af')
return_id: 999
order_id: 4321
payment_id: 123456
email_id: "suresh@gmail.com"
eligibility: "True"
return_payment_details: Array
  0: Object
    cart_type: "Credit"
    card_number: 123456
    exp_date: "3-3-2029"
    Name: "Suresh Garimella"
return_status: "delivered"
time_stamp: "4-2-2023"
```

Admin-



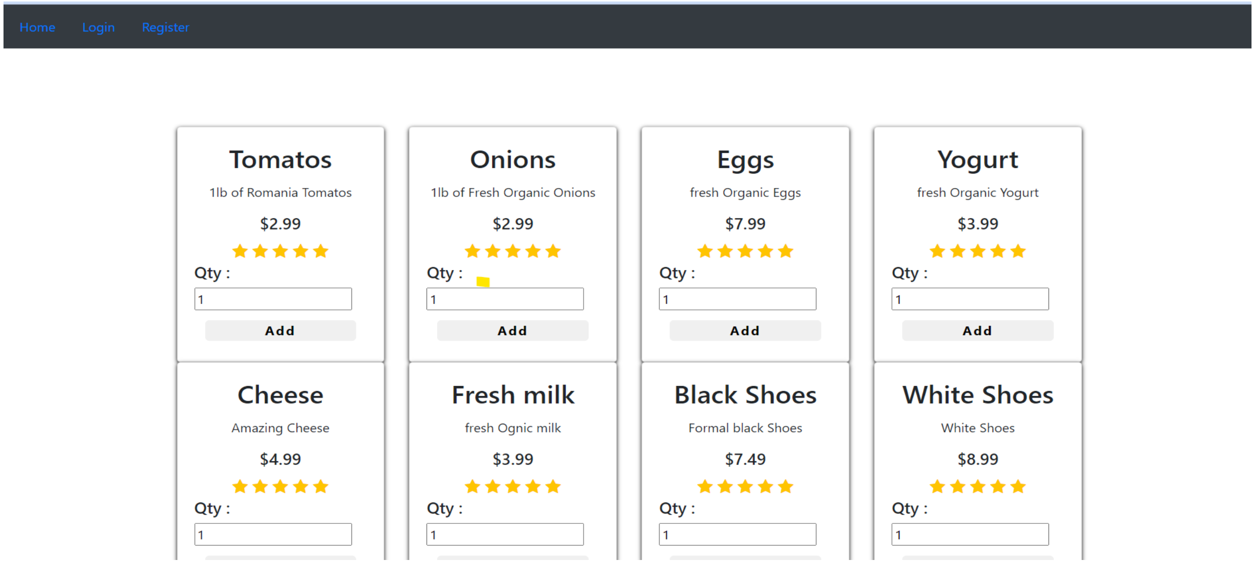
```
_id: ObjectId('64160f1e35b9f34d6e7eeb69')
name: "Admin FirstName and LasrtName"
username: "admin@gmail.com"
password: "password"
phno: 9131234567
```

Summary :

This project helped us to learn most of the real time experiences on mongodb and python and html bootstrap . Most of the features of online ecommerce websites are implemented here. We look forward to implementing further changes as per future scope.

User interface and forms

Home page :



Tomatos
1lb of Romania Tomatos
$2.99
★★★★★
Qty : 1

Onions
1lb of Fresh Organic Onions
$2.99
★★★★★
Qty : 1

Eggs
fresh Organic Eggs
$7.99
★★★★★
Qty : 1

Yogurt
fresh Organic Yogurt
$3.99
★★★★★
Qty : 1

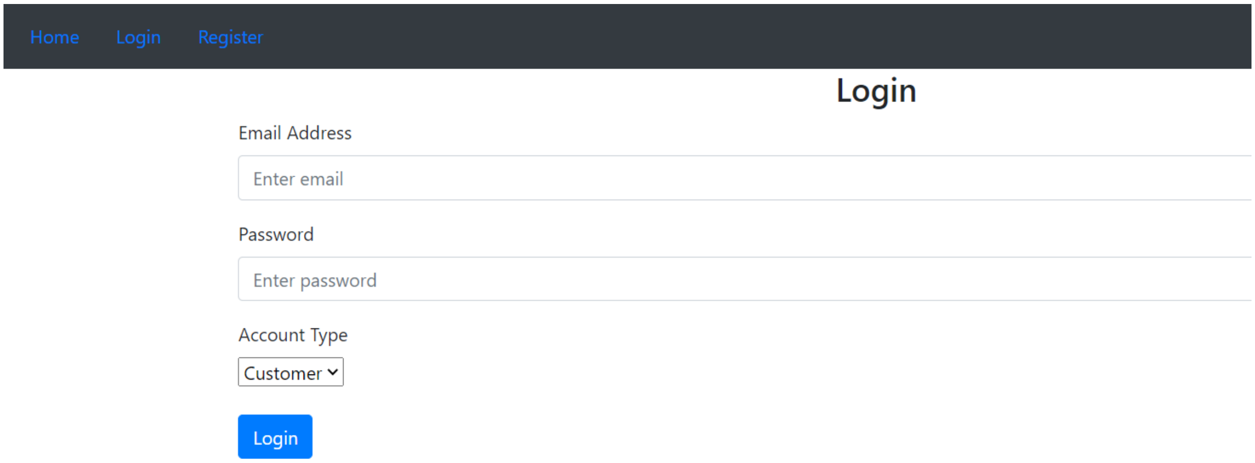
Cheese
Amazing Cheese
$4.99
★★★★★
Qty : 1

Fresh milk
fresh Ognic milk
$3.99
★★★★★
Qty : 1

Black Shoes
Formal black Shoes
$7.49
★★★★★
Qty : 1

White Shoes
White Shoes
$8.99
★★★★★
Qty : 1

Admin / customer Login Page:



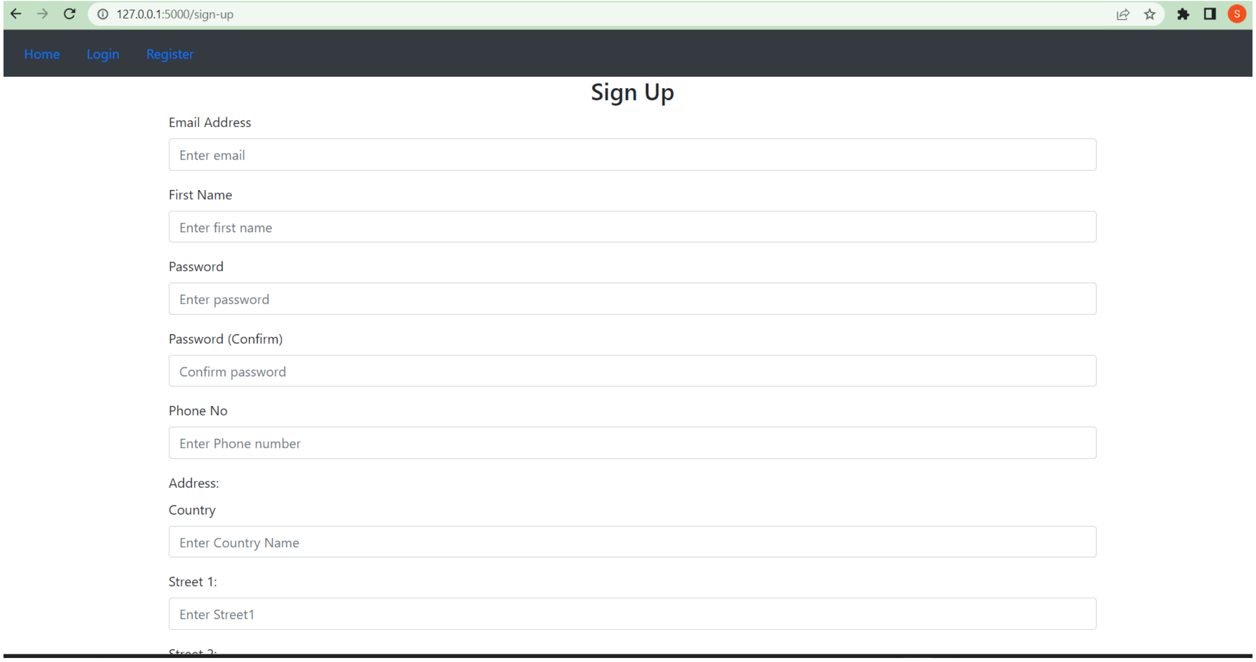
Login

Email Address
Enter email

Password
Enter password

Account Type
Customer

Customer Registration Page:-



Sign Up

Email Address

Enter email

First Name

Enter first name

Password

Enter password

Password (Confirm)

Confirm password

Phone No

Enter Phone number

Address:

Country

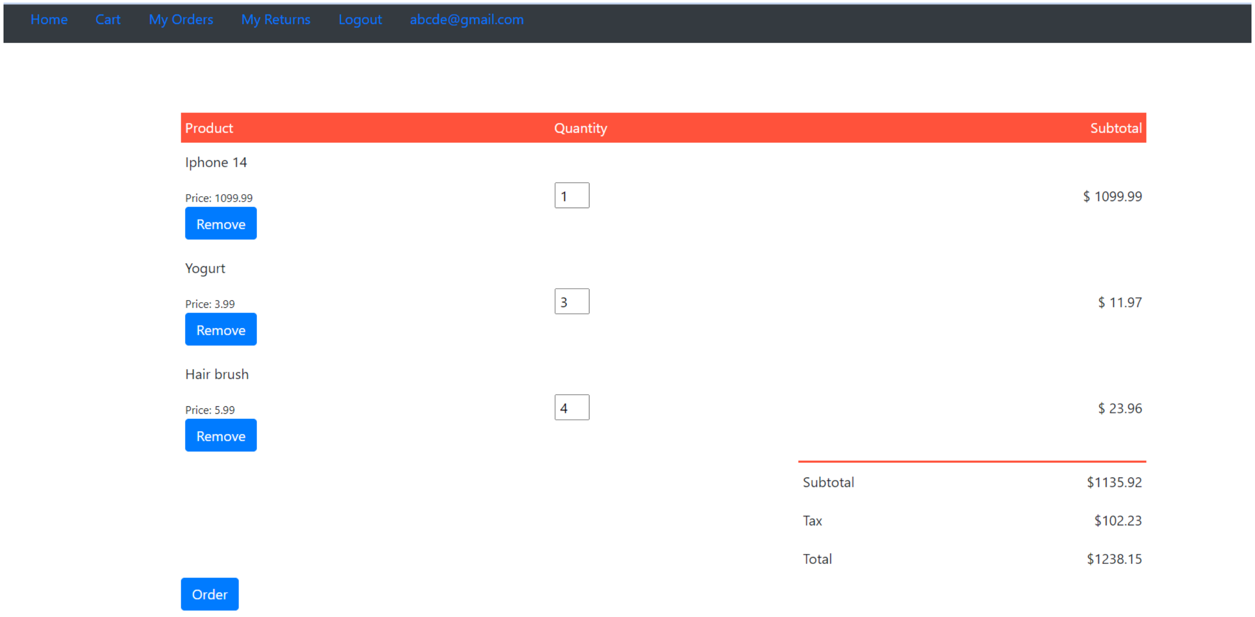
Enter Country Name

Street 1:

Enter Street1

Street 2:

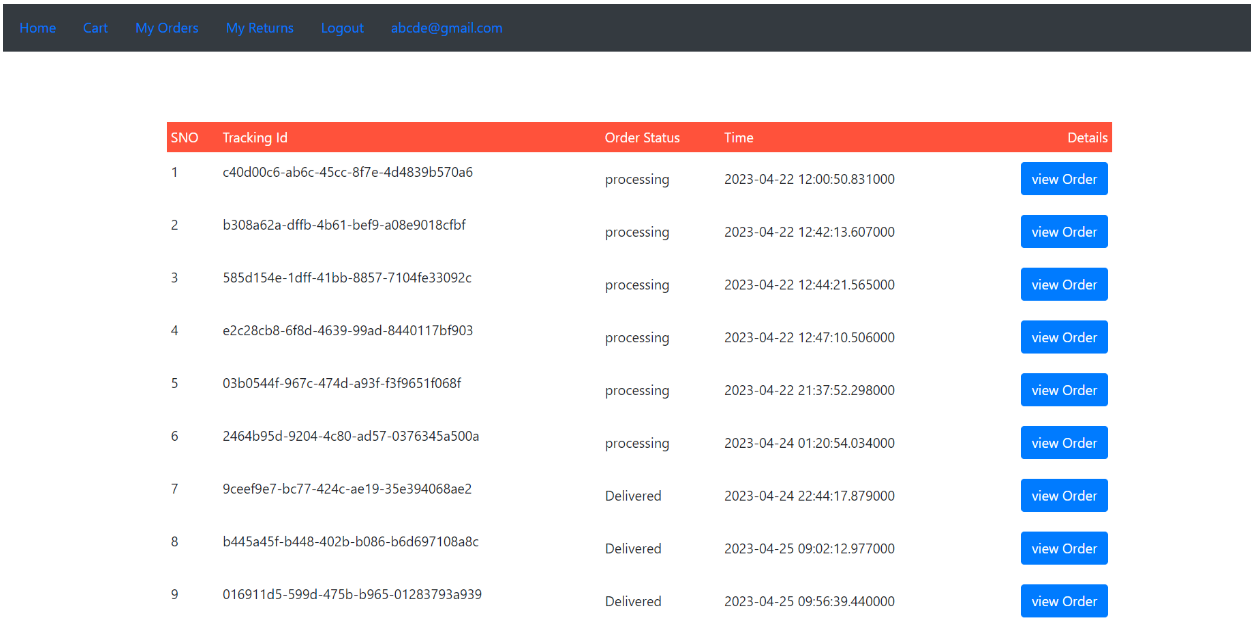
User cart_page:-



Product	Quantity	Subtotal
Iphone 14 Price: 1099.99	1	$ 1099.99
Yogurt Price: 3.99	3	$ 11.97
Hair brush Price: 5.99	4	$ 23.96

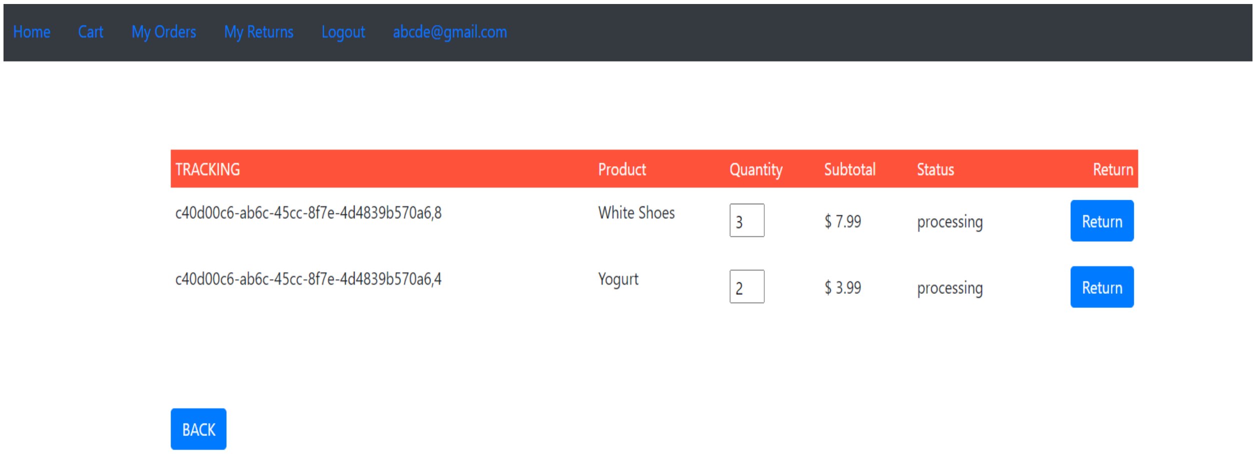
Subtotal	$1135.92
Tax	$102.23
Total	$1238.15

User All orders page:-



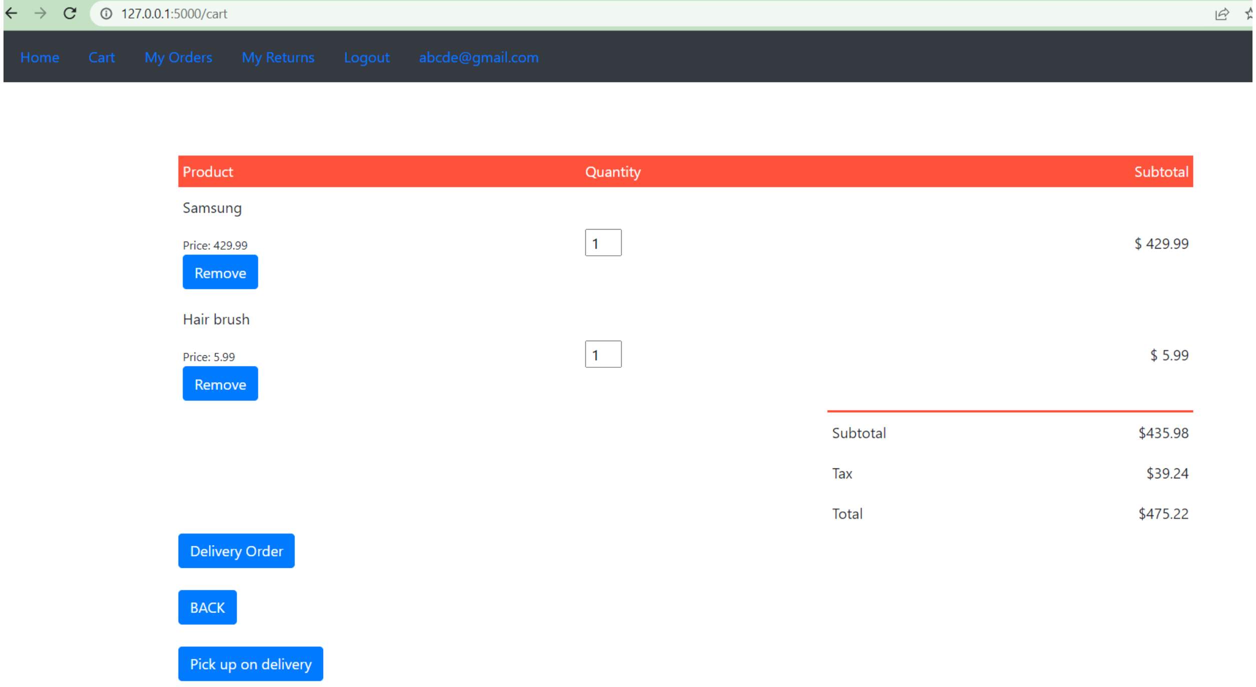
SNO	Tracking Id	Order Status	Time	Details
1	c40d00c6-ab6c-45cc-8f7e-4d4839b570a6	processing	2023-04-22 12:00:50.831000	view Order
2	b308a62a-dffb-4b61-bef9-a08e9018cfbf	processing	2023-04-22 12:42:13.607000	view Order
3	585d154e-1dff-41bb-8857-7104fe33092c	processing	2023-04-22 12:44:21.565000	view Order
4	e2c28cb8-6f8d-4639-99ad-8440117bf903	processing	2023-04-22 12:47:10.506000	view Order
5	03b0544f-967c-474d-a93f-f3f9651f068f	processing	2023-04-22 21:37:52.298000	view Order
6	2464b95d-9204-4c80-ad57-0376345a500a	processing	2023-04-24 01:20:54.034000	view Order
7	9ceef9e7-bc77-424c-ae19-35e394068ae2	Delivered	2023-04-24 22:44:17.879000	view Order
8	b445a45f-b448-402b-b086-b6d697108a8c	Delivered	2023-04-25 09:02:12.977000	view Order
9	016911d5-599d-475b-b965-01283793a939	Delivered	2023-04-25 09:56:39.440000	view Order

User view Individual Orders page:-



TRACKING	Product	Quantity	Subtotal	Status	Return
c40d00c6-ab6c-45cc-8f7e-4d4839b570a6,8	White Shoes	3	$ 7.99	processing	Return
c40d00c6-ab6c-45cc-8f7e-4d4839b570a6,4	Yogurt	2	$ 3.99	processing	Return

User Check-Out Page:-

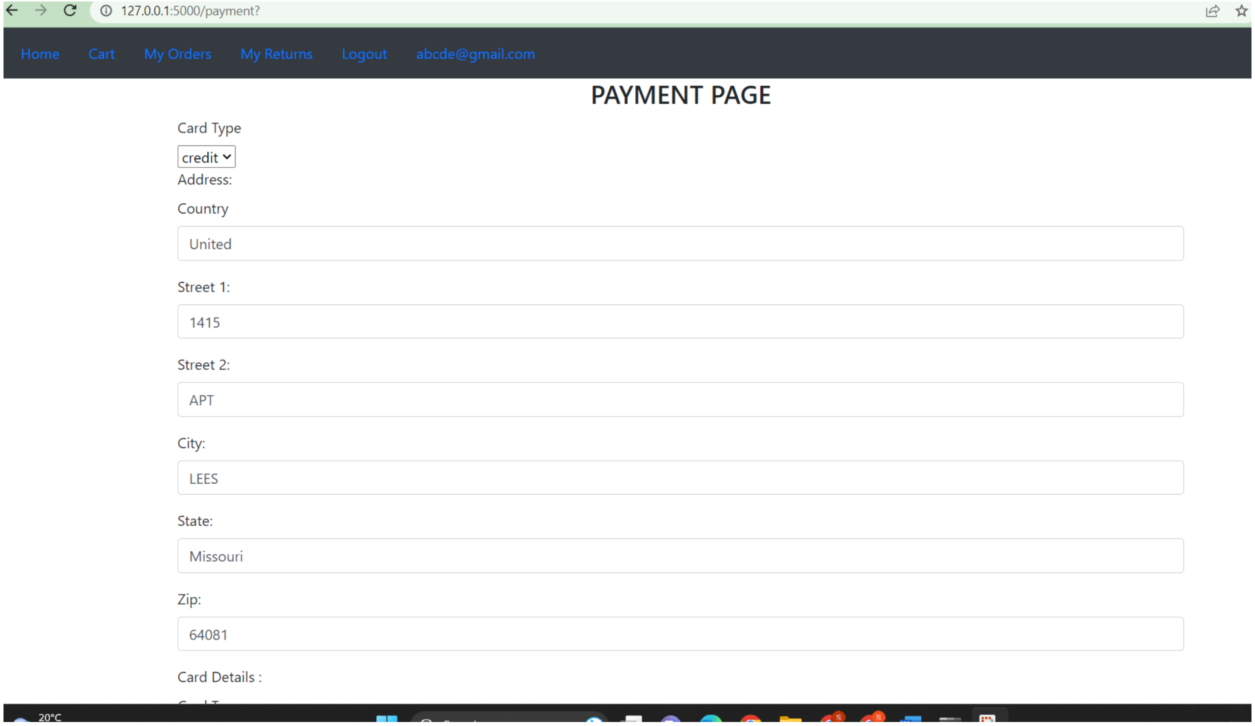


127.0.0.1:5000/cart

Product	Quantity	Subtotal
Samsung Price: 429.99 Remove	1	$ 429.99
Hair brush Price: 5.99 Remove	1	$ 5.99

Subtotal	$435.98
Tax	$39.24
Total	$475.22

Customer Payment Page:-



127.0.0.1:5000/payment?

PAYMENT PAGE

Card Type

credit

Address:

Country: United

Street 1: 1415

Street 2: APT

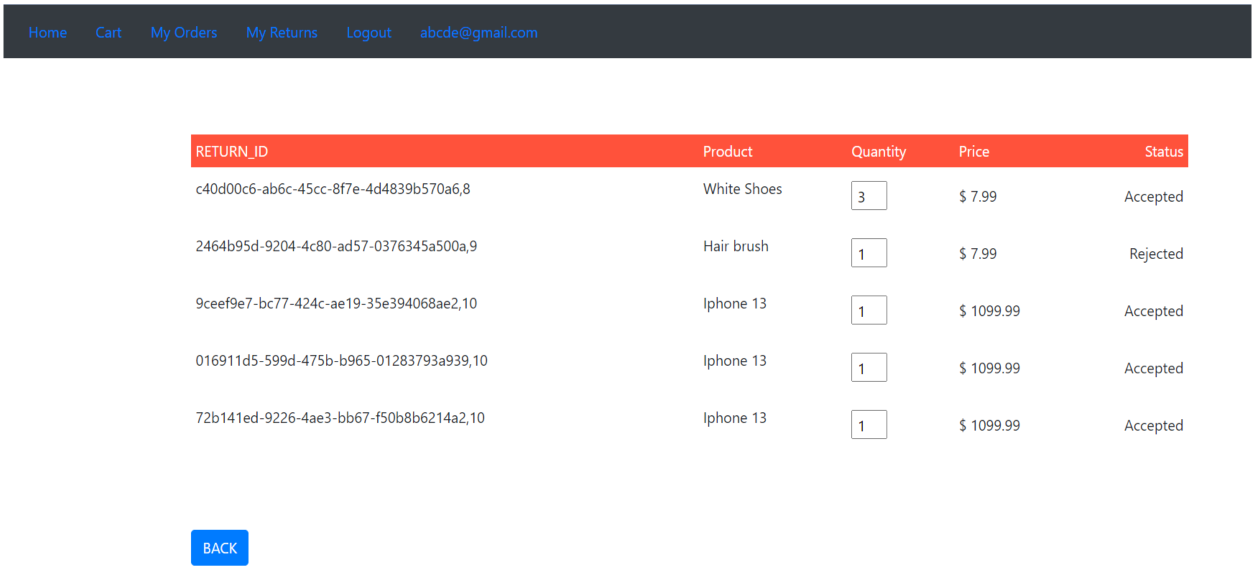
City: LEES

State: Missouri

Zip: 64081

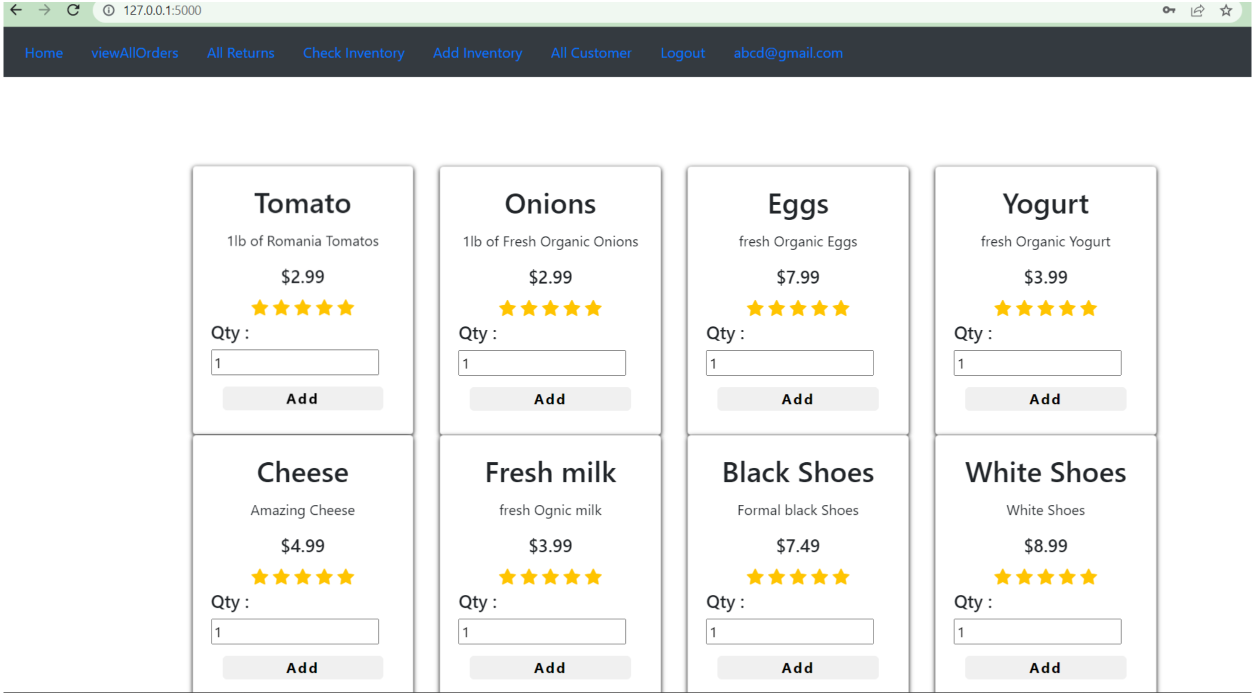
Card Details :

User Order Returns Page:-



RETURN_ID	Product	Quantity	Price	Status
c40d00c6-ab6c-45cc-8f7e-4d4839b570a6,8	White Shoes	3	$ 7.99	Accepted
2464b95d-9204-4c80-ad57-0376345a500a,9	Hair brush	1	$ 7.99	Rejected
9ceef9e7-bc77-424c-ae19-35e394068ae2,10	Iphone 13	1	$ 1099.99	Accepted
016911d5-599d-475b-b965-01283793a939,10	Iphone 13	1	$ 1099.99	Accepted
72b141ed-9226-4ae3-bb67-f50b8b6214a2,10	Iphone 13	1	$ 1099.99	Accepted

Admin Home page:-



Tomato
1lb of Romania Tomatos
$2.99
★★★★★
Qty :
1
Add

Onions
1lb of Fresh Organic Onions
$2.99
★★★★★
Qty :
1
Add

Eggs
fresh Organic Eggs
$7.99
★★★★★
Qty :
1
Add

Yogurt
fresh Organic Yogurt
$3.99
★★★★★
Qty :
1
Add

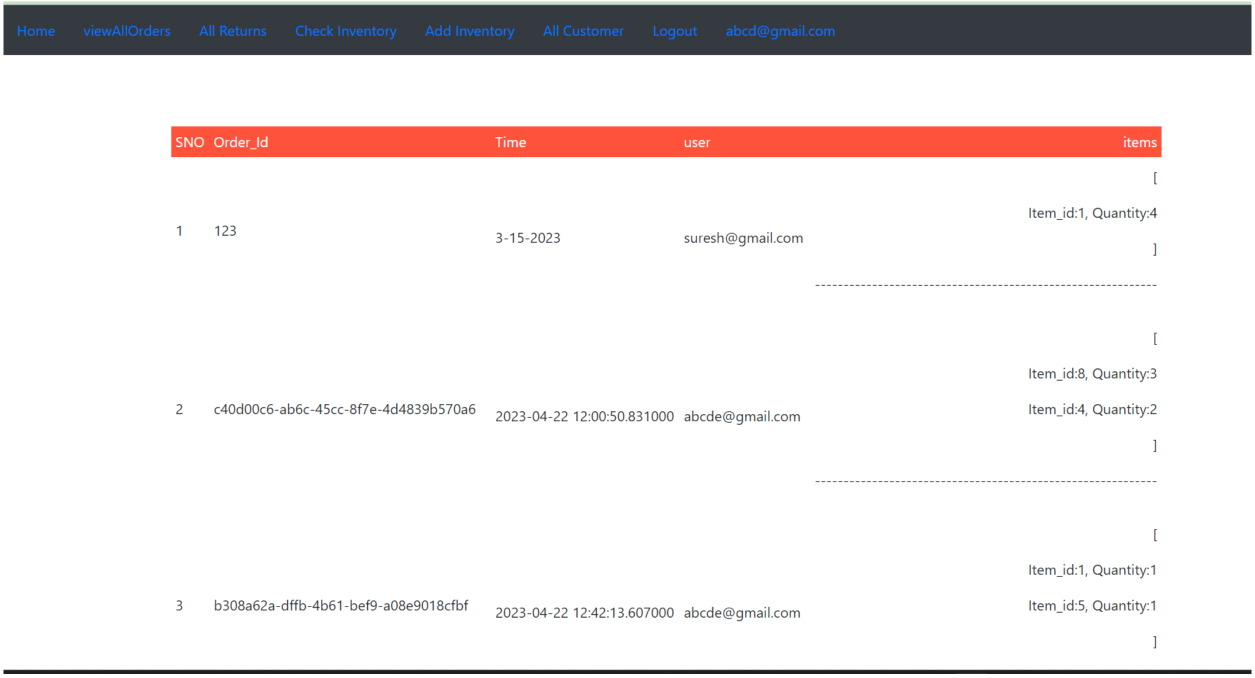
Cheese
Amazing Cheese
$4.99
★★★★★
Qty :
1
Add

Fresh milk
fresh Ognic milk
$3.99
★★★★★
Qty :
1
Add

Black Shoes
Formal black Shoes
$7.49
★★★★★
Qty :
1
Add

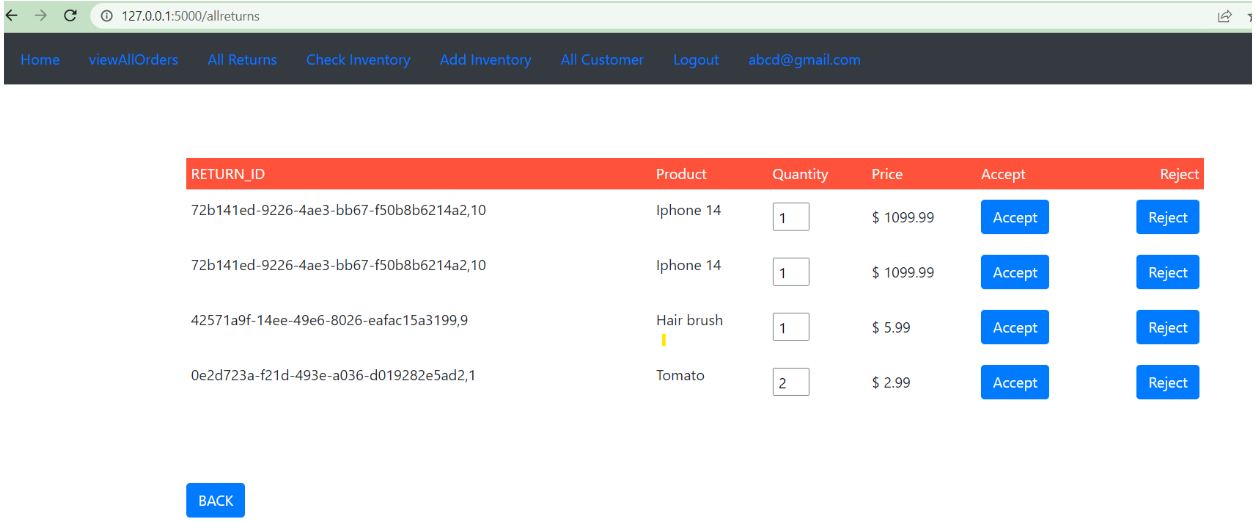
White Shoes
White Shoes
$8.99
★★★★★
Qty :
1
Add

Admin View All Orders page:-



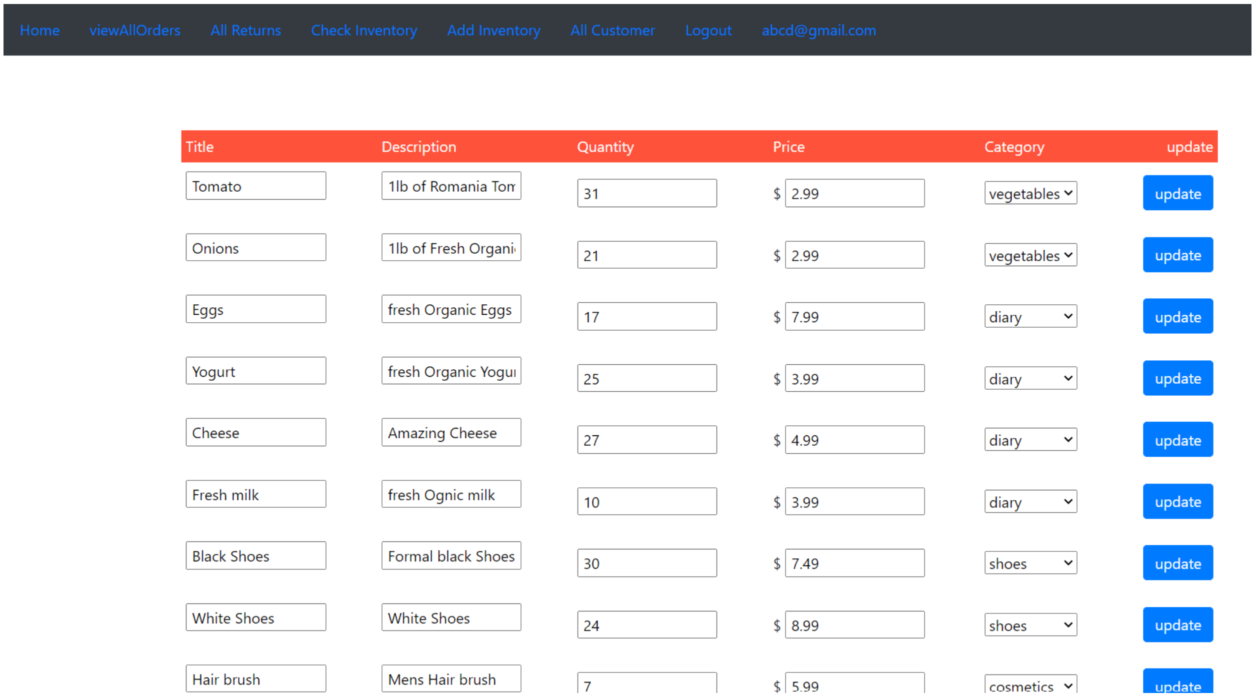
SNO	Order_Id	Time	user	items
1	123	3-15-2023	suresh@gmail.com	[Item_id:1, Quantity:4] --
2	c40d00c6-ab6c-45cc-8f7e-4d4839b570a6	2023-04-22 12:00:50.831000	abcde@gmail.com	[Item_id:8, Quantity:3 Item_id:4, Quantity:2] --
3	b308a62a-dffb-4b61-bef9-a08e9018cfbf	2023-04-22 12:42:13.607000	abcde@gmail.com	[Item_id:1, Quantity:1 Item_id:5, Quantity:1]

Admin View All Returns(Accept / Reject) Page:-



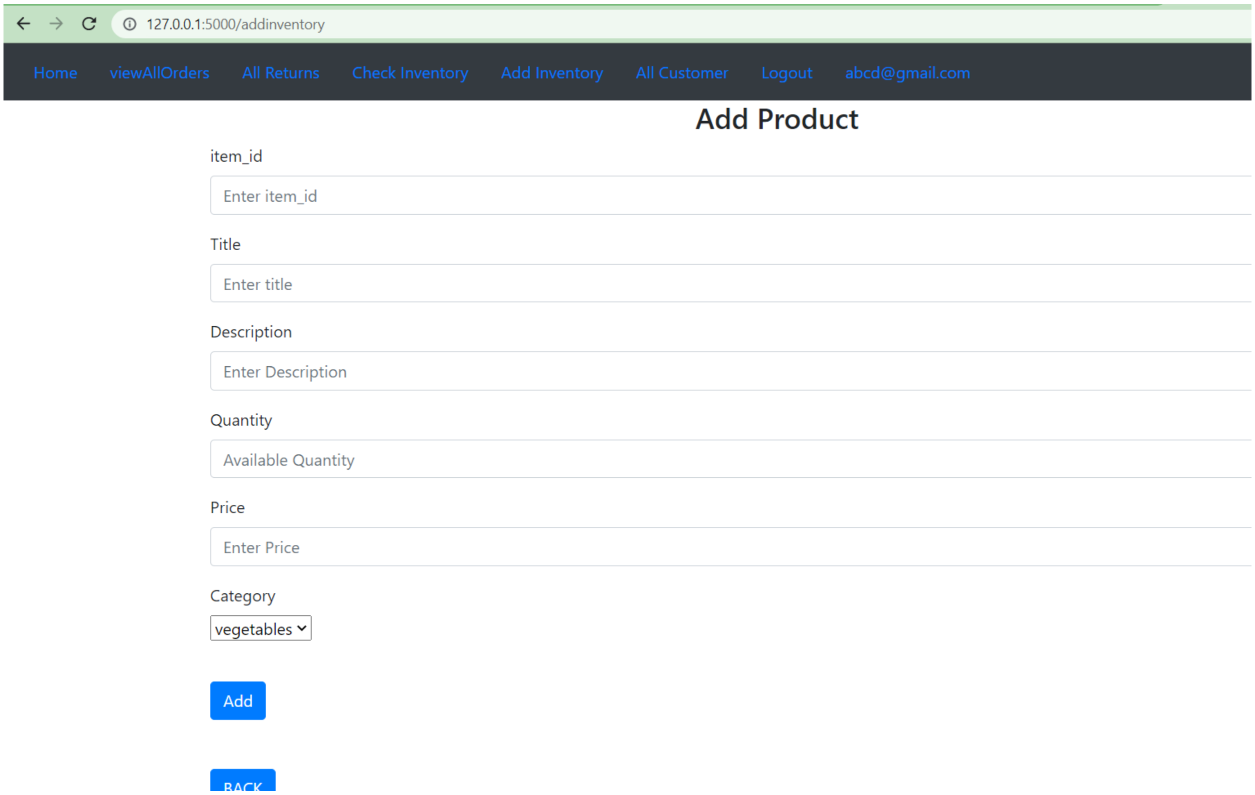
RETURN_ID	Product	Quantity	Price	Accept	Reject
72b141ed-9226-4ae3-bb67-f50b8b6214a2,10	Iphone 14	1	$ 1099.99	Accept	Reject
72b141ed-9226-4ae3-bb67-f50b8b6214a2,10	Iphone 14	1	$ 1099.99	Accept	Reject
42571a9f-14ee-49e6-8026-eafac15a3199,9	Hair brush	1	$ 5.99	Accept	Reject
0e2d723a-f21d-493e-a036-d019282e5ad2,1	Tomato	2	$ 2.99	Accept	Reject

Admin view and Update Inventory page:-



Title	Description	Quantity	Price	Category	update
Tomato	1lb of Romania Tom	31	$ 2.99	vegetables	update
Onions	1lb of Fresh Organi	21	$ 2.99	vegetables	update
Eggs	fresh Organic Eggs	17	$ 7.99	diary	update
Yogurt	fresh Organic Yogu	25	$ 3.99	diary	update
Cheese	Amazing Cheese	27	$ 4.99	diary	update
Fresh milk	fresh Ognic milk	10	$ 3.99	diary	update
Black Shoes	Formal black Shoes	30	$ 7.49	shoes	update
White Shoes	White Shoes	24	$ 8.99	shoes	update
Hair brush	Mens Hair brush	7	$ 5.99	cosmetics	update

Admin Add Inventory Page:-



Add Product

item_id

Enter item_id

Title

Enter title

Description

Enter Description

Quantity

Available Quantity

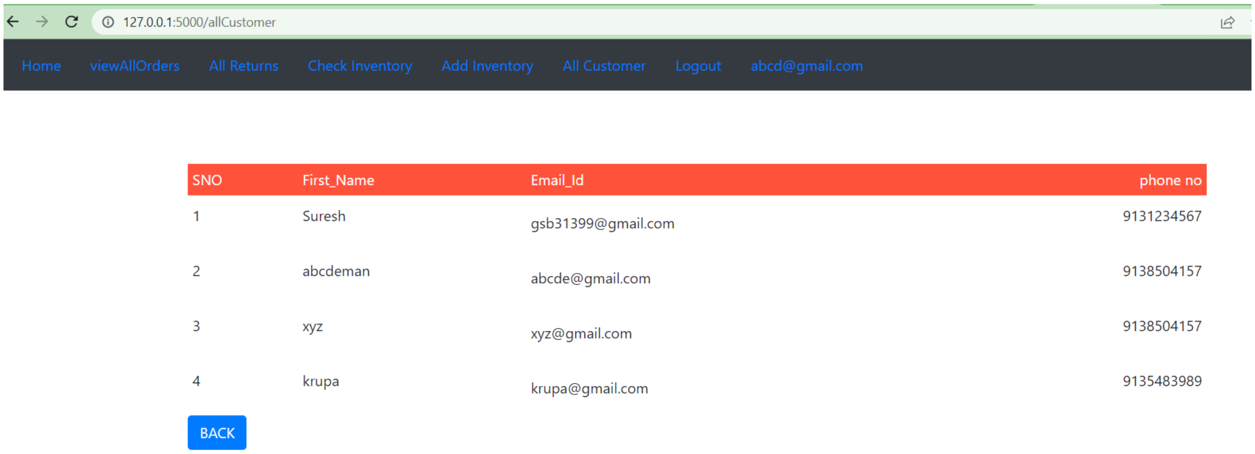
Price

Enter Price

Category

vegetables

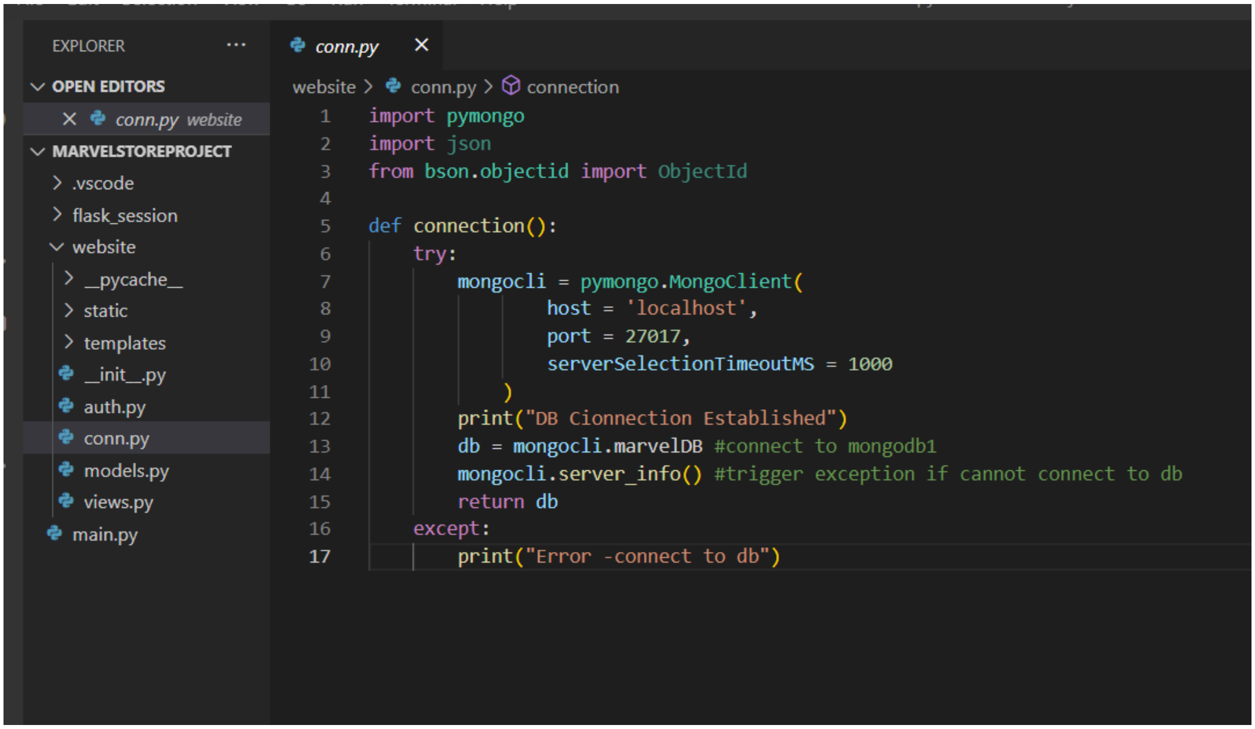
View All Customers Page:-



SNO	First_Name	Email_Id	phone no
1	Suresh	gsb31399@gmail.com	9131234567
2	abcdeman	abcde@gmail.com	9138504157
3	xyz	xyz@gmail.com	9138504157
4	krupa	krupa@gmail.com	9135483989

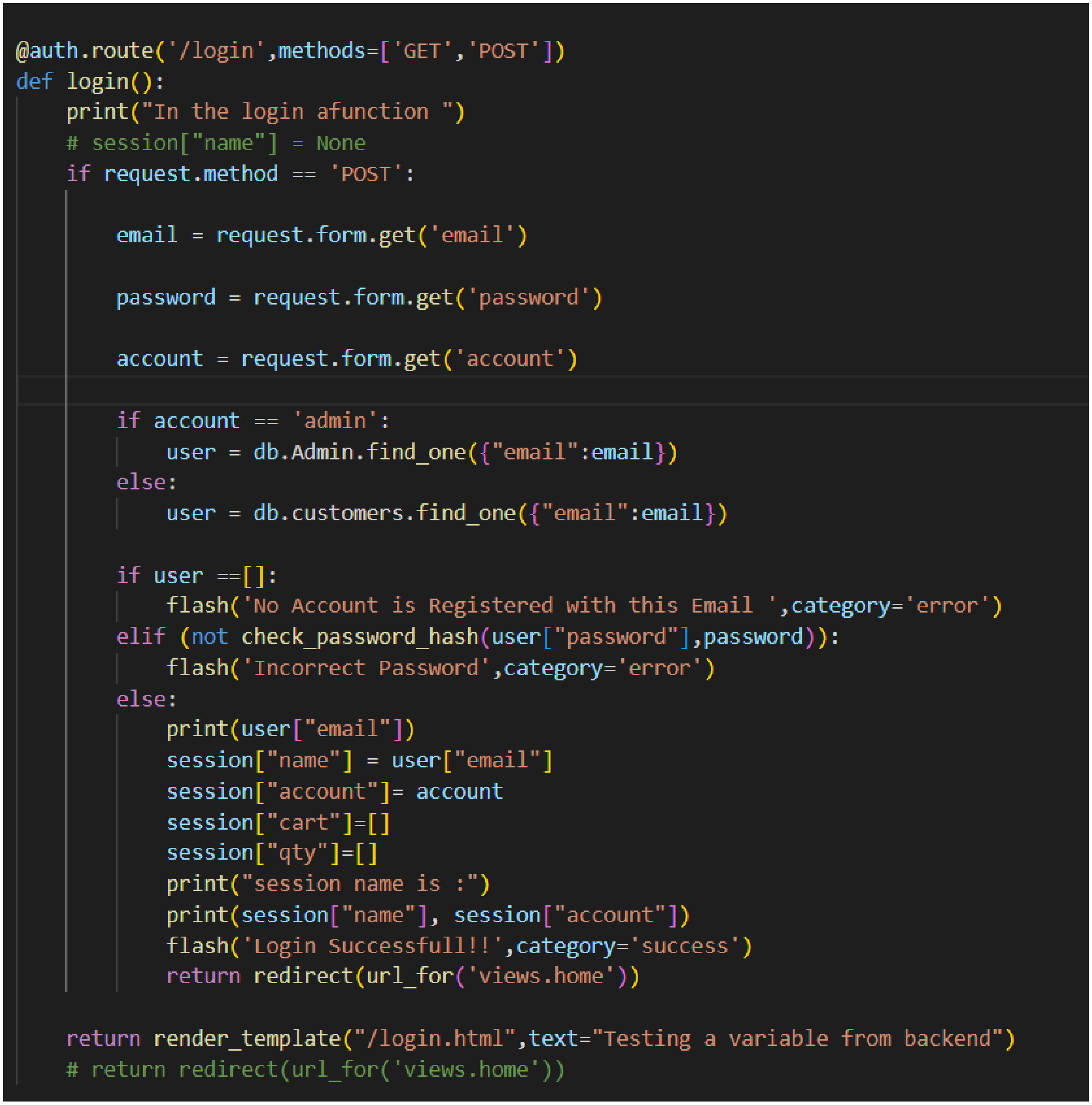
Source code :

Mongodb connection logic



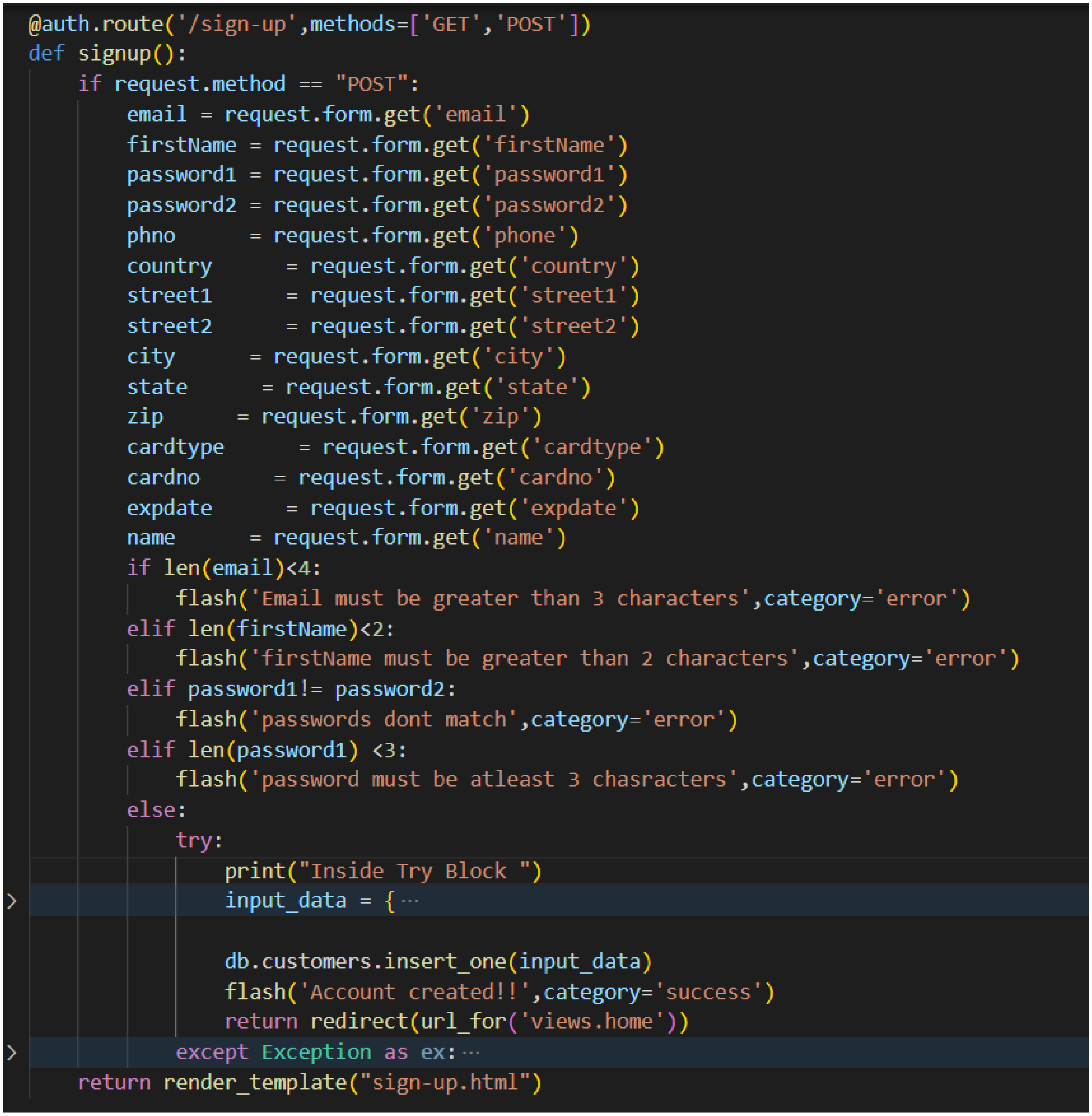
```python
import pymongo
import json
from bson.objectid import ObjectId

def connection():
    try:
        mongocli = pymongo.MongoClient(
                host = 'localhost',
                port = 27017,
                serverSelectionTimeoutMS = 1000
            )
        print("DB Cionnection Established")
        db = mongocli.marvelDB #connect to mongodb1
        mongocli.server_info() #trigger exception if cannot connect to db
        return db
    except:
        print("Error -connect to db")
```

Login page code snippet :-



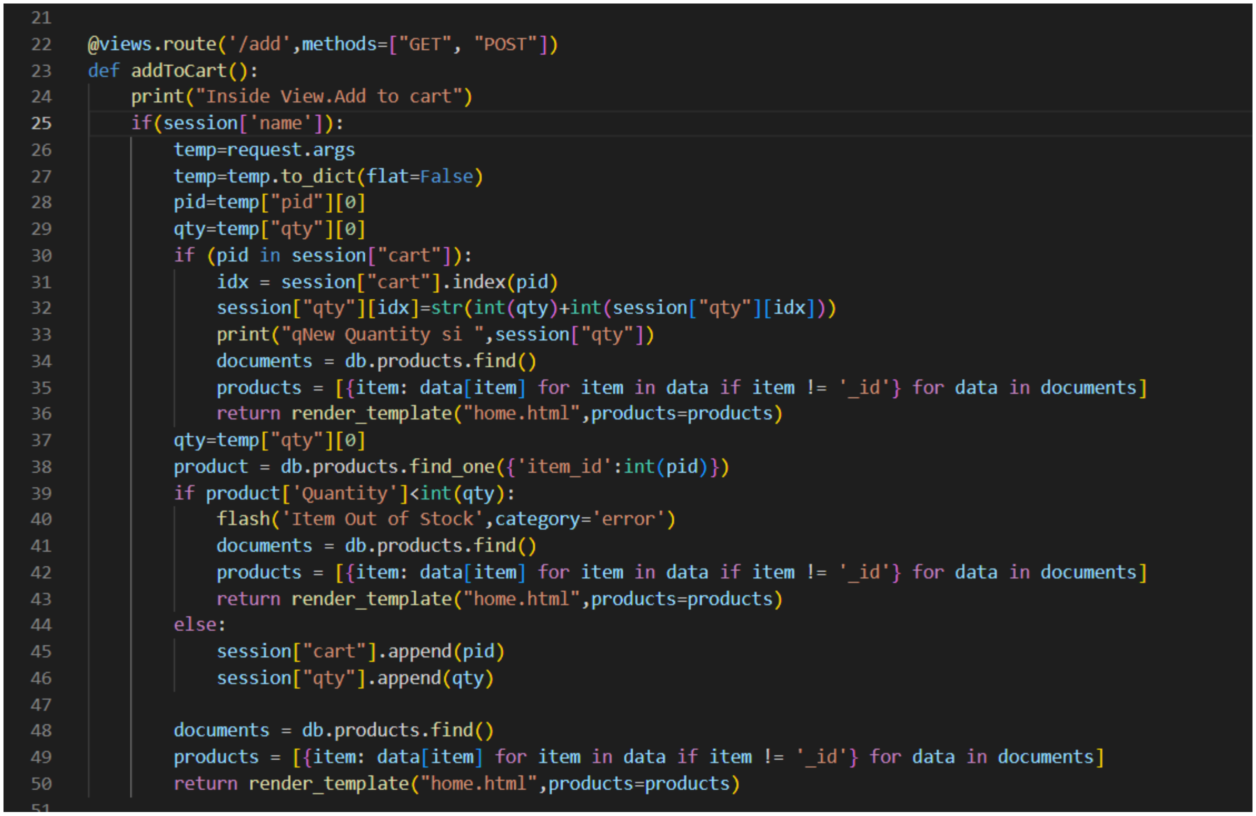
```
@auth.route('/login',methods=['GET','POST'])
def login():
    print("In the login afunction ")
    # session["name"] = None
    if request.method == 'POST':

        email = request.form.get('email')

        password = request.form.get('password')

        account = request.form.get('account')

        if account == 'admin':
            user = db.Admin.find_one({"email":email})
        else:
            user = db.customers.find_one({"email":email})

        if user ==[]:
            flash('No Account is Registered with this Email ',category='error')
        elif (not check_password_hash(user["password"],password)):
            flash('Incorrect Password',category='error')
        else:
            print(user["email"])
            session["name"] = user["email"]
            session["account"]= account
            session["cart"]=[]
            session["qty"]=[]
            print("session name is :")
            print(session["name"], session["account"])
            flash('Login Successfull!!',category='success')
            return redirect(url_for('views.home'))

    return render_template("/login.html",text="Testing a variable from backend")
    # return redirect(url_for('views.home'))
```

Sign-Up logic:-



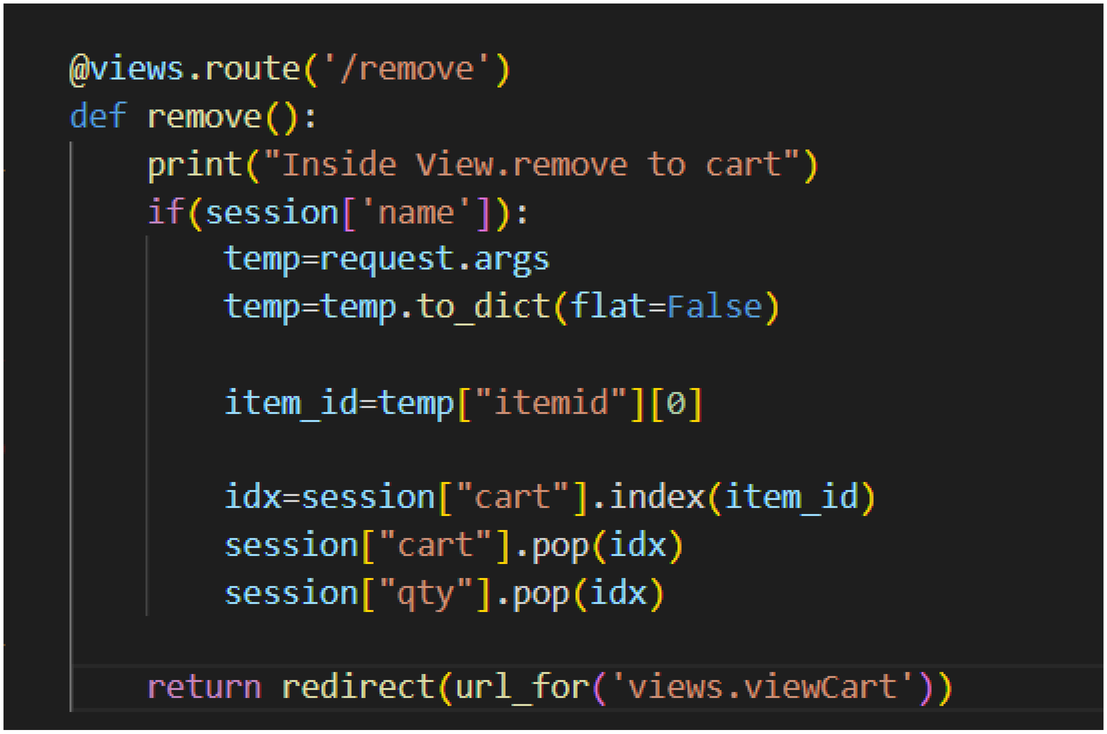
```
@auth.route('/sign-up',methods=['GET','POST'])
def signup():
    if request.method == "POST":
        email = request.form.get('email')
        firstName = request.form.get('firstName')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')
        phno      = request.form.get('phone')
        country      = request.form.get('country')
        street1      = request.form.get('street1')
        street2      = request.form.get('street2')
        city      = request.form.get('city')
        state      = request.form.get('state')
        zip      = request.form.get('zip')
        cardtype      = request.form.get('cardtype')
        cardno      = request.form.get('cardno')
        expdate      = request.form.get('expdate')
        name      = request.form.get('name')
        if len(email)<4:
            flash('Email must be greater than 3 characters',category='error')
        elif len(firstName)<2:
            flash('firstName must be greater than 2 characters',category='error')
        elif password1!= password2:
            flash('passwords dont match',category='error')
        elif len(password1) <3:
            flash('password must be atleast 3 chasracters',category='error')
        else:
            try:
                print("Inside Try Block ")
                input_data = {···

                db.customers.insert_one(input_data)
                flash('Account created!!',category='success')
                return redirect(url_for('views.home'))
            except Exception as ex:···
    return render_template("sign-up.html")
```

Cart logic:-



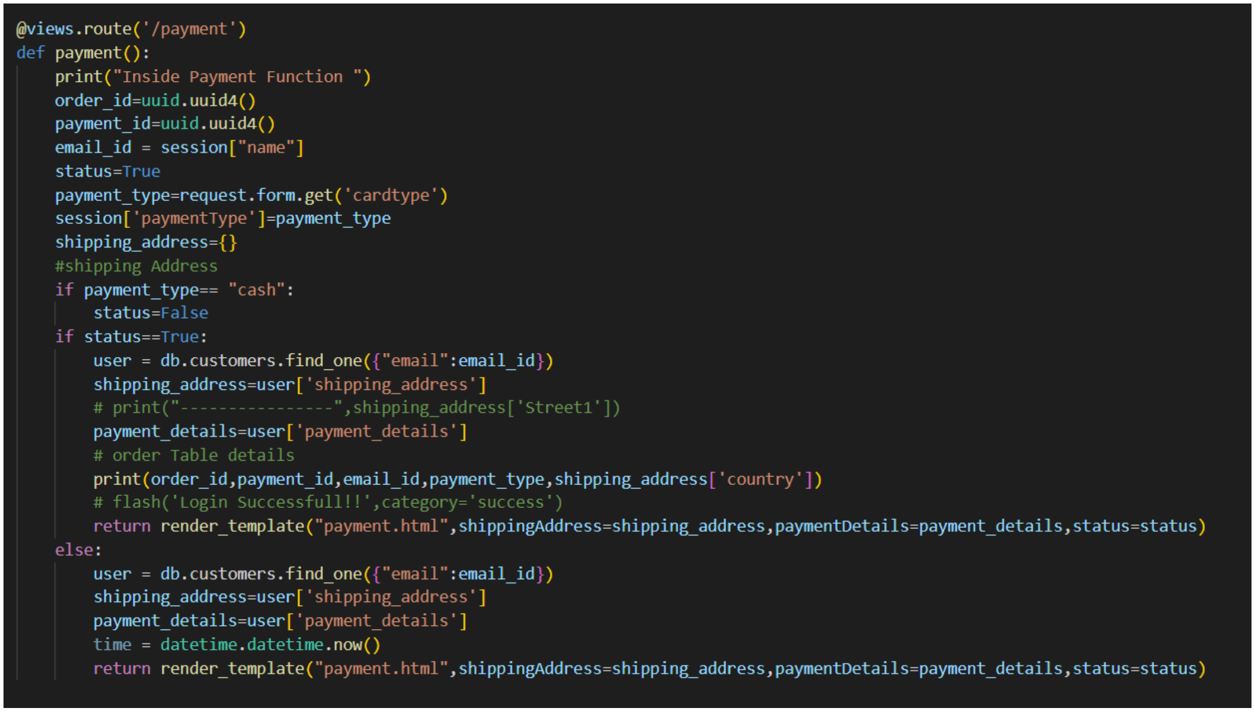
```python
@views.route('/add',methods=["GET", "POST"])
def addToCart():
    print("Inside View.Add to cart")
    if(session['name']):
        temp=request.args
        temp=temp.to_dict(flat=False)
        pid=temp["pid"][0]
        qty=temp["qty"][0]
        if (pid in session["cart"]):
            idx = session["cart"].index(pid)
            session["qty"][idx]=str(int(qty)+int(session["qty"][idx]))
            print("qNew Quantity si ",session["qty"])
            documents = db.products.find()
            products = [{item: data[item] for item in data if item != '_id'} for data in documents]
            return render_template("home.html",products=products)
        qty=temp["qty"][0]
        product = db.products.find_one({'item_id':int(pid)})
        if product['Quantity']<int(qty):
            flash('Item Out of Stock',category='error')
            documents = db.products.find()
            products = [{item: data[item] for item in data if item != '_id'} for data in documents]
            return render_template("home.html",products=products)
        else:
            session["cart"].append(pid)
            session["qty"].append(qty)

        documents = db.products.find()
        products = [{item: data[item] for item in data if item != '_id'} for data in documents]
        return render_template("home.html",products=products)
```

Remove product from cart logic:-



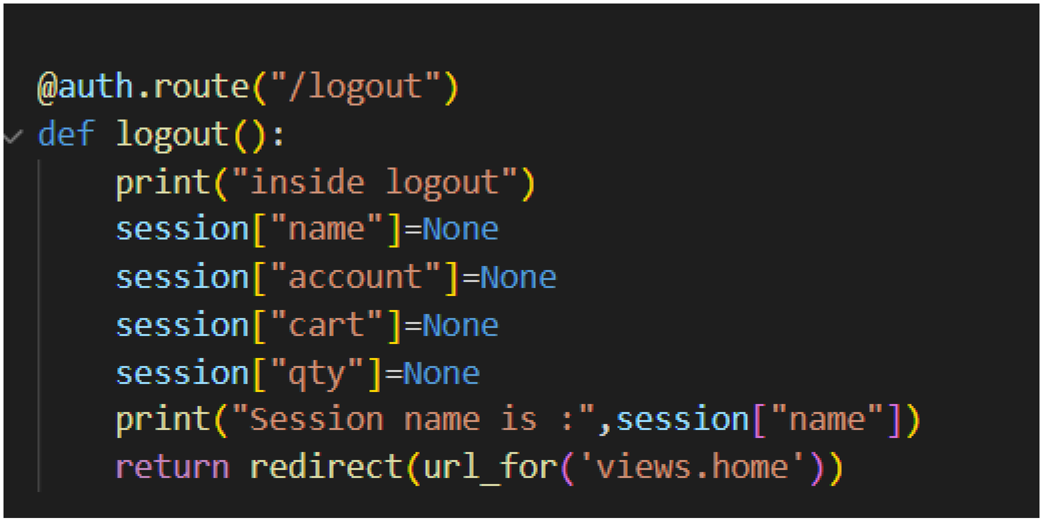
```python
@views.route('/remove')
def remove():
    print("Inside View.remove to cart")
    if(session['name']):
        temp=request.args
        temp=temp.to_dict(flat=False)

        item_id=temp["itemid"][0]

        idx=session["cart"].index(item_id)
        session["cart"].pop(idx)
        session["qty"].pop(idx)

    return redirect(url_for('views.viewCart'))
```

Remove product from cart logic:-



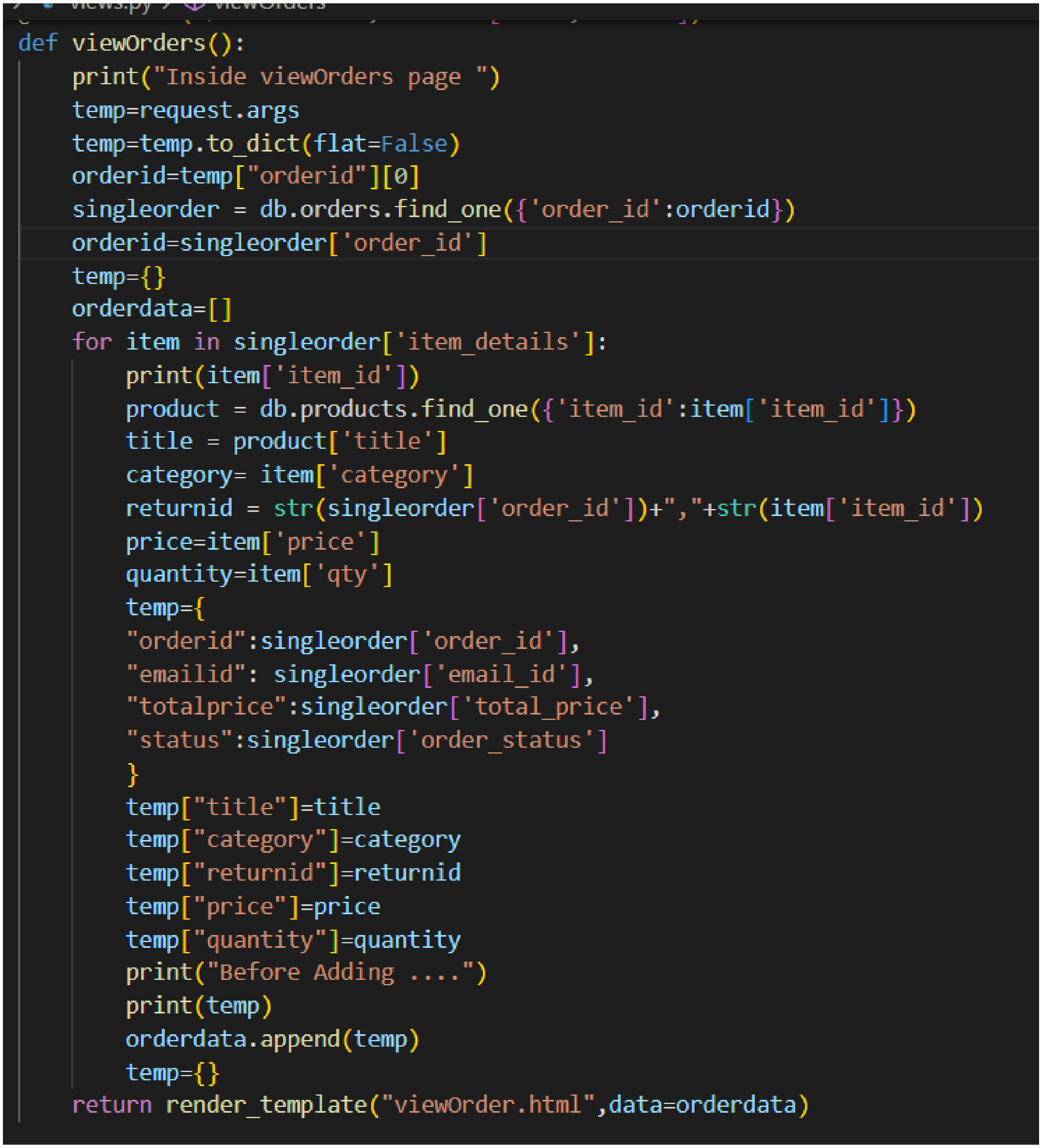
```python
@views.route('/payment')
def payment():
    print("Inside Payment Function ")
    order_id=uuid.uuid4()
    payment_id=uuid.uuid4()
    email_id = session["name"]
    status=True
    payment_type=request.form.get('cardtype')
    session['paymentType']=payment_type
    shipping_address={}
    #shipping Address
    if payment_type== "cash":
        status=False
    if status==True:
        user = db.customers.find_one({"email":email_id})
        shipping_address=user['shipping_address']
        # print("----------------",shipping_address['Street1'])
        payment_details=user['payment_details']
        # order Table details
        print(order_id,payment_id,email_id,payment_type,shipping_address['country'])
        # flash('Login Successfull!!',category='success')
        return render_template("payment.html",shippingAddress=shipping_address,paymentDetails=payment_details,status=status)
    else:
        user = db.customers.find_one({"email":email_id})
        shipping_address=user['shipping_address']
        payment_details=user['payment_details']
        time = datetime.datetime.now()
        return render_template("payment.html",shippingAddress=shipping_address,paymentDetails=payment_details,status=status)
```

LogOut Code:-



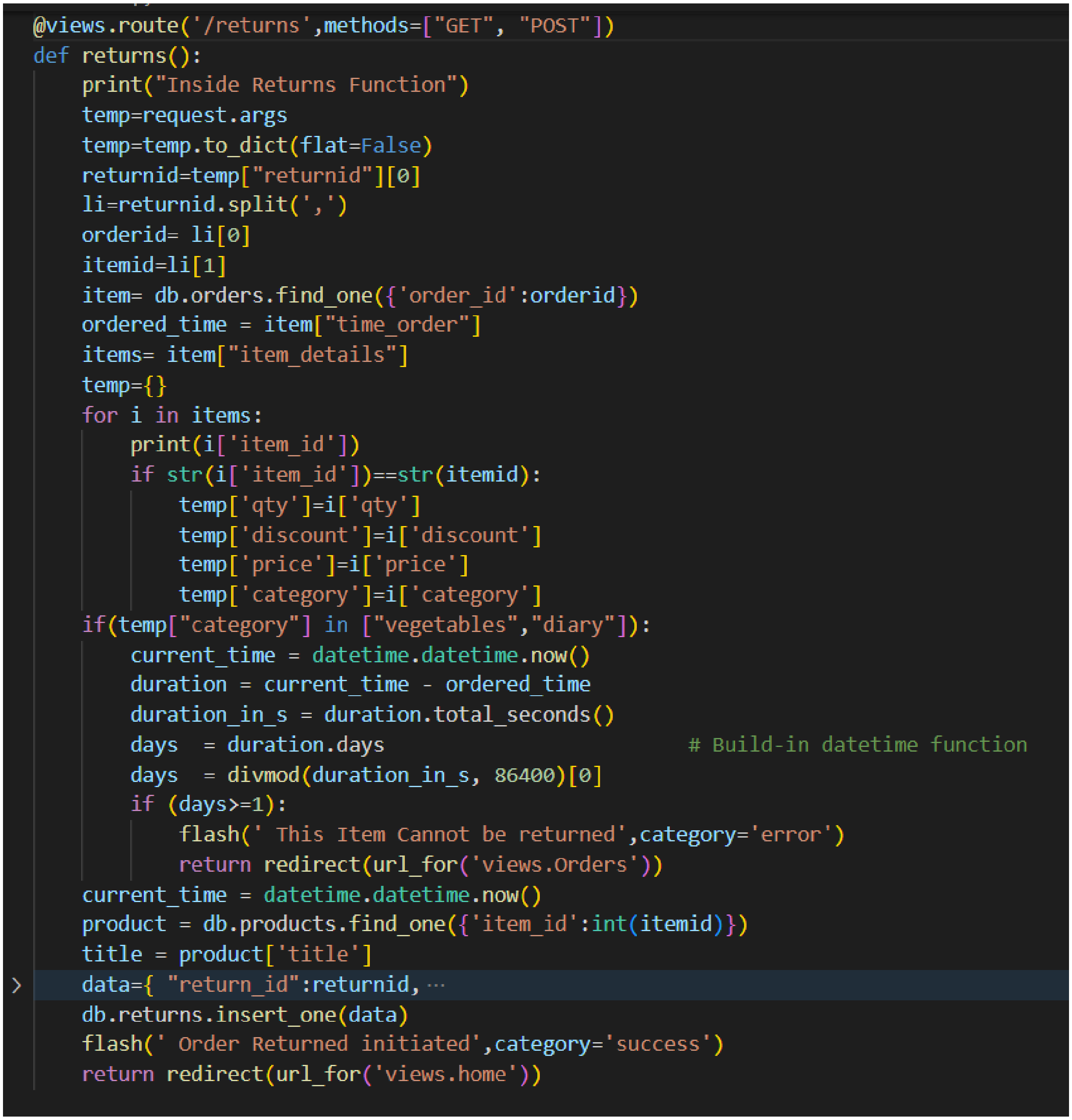
```python
@auth.route("/logout")
def logout():
    print("inside logout")
    session["name"]=None
    session["account"]=None
    session["cart"]=None
    session["qty"]=None
    print("Session name is :",session["name"])
    return redirect(url_for('views.home'))
```

View Order Code:-



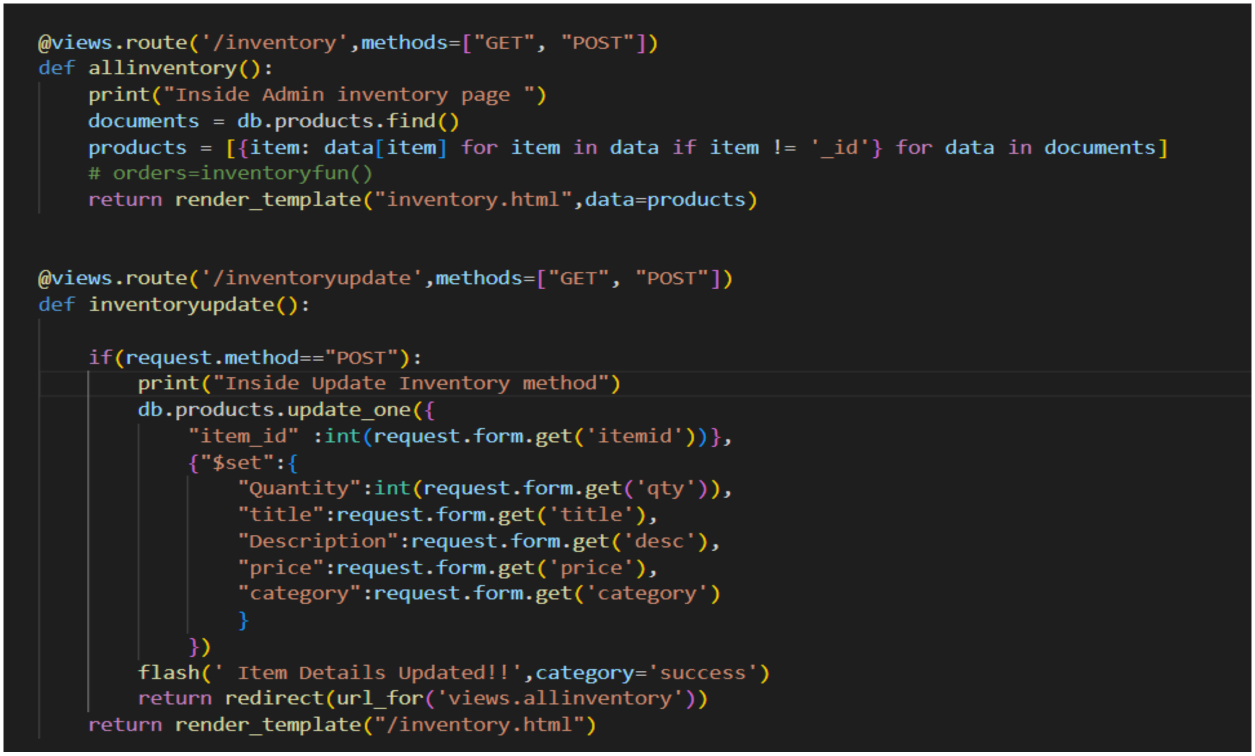
```
def viewOrders():
    print("Inside viewOrders page ")
    temp=request.args
    temp=temp.to_dict(flat=False)
    orderid=temp["orderid"][0]
    singleorder = db.orders.find_one({'order_id':orderid})
    orderid=singleorder['order_id']
    temp={}
    orderdata=[]
    for item in singleorder['item_details']:
        print(item['item_id'])
        product = db.products.find_one({'item_id':item['item_id']})
        title = product['title']
        category= item['category']
        returnid = str(singleorder['order_id'])+","+str(item['item_id'])
        price=item['price']
        quantity=item['qty']
        temp={
        "orderid":singleorder['order_id'],
        "emailid": singleorder['email_id'],
        "totalprice":singleorder['total_price'],
        "status":singleorder['order_status']
        }
        temp["title"]=title
        temp["category"]=category
        temp["returnid"]=returnid
        temp["price"]=price
        temp["quantity"]=quantity
        print("Before Adding ....")
        print(temp)
        orderdata.append(temp)
        temp={}
    return render_template("viewOrder.html",data=orderdata)
```

Returns Code:-



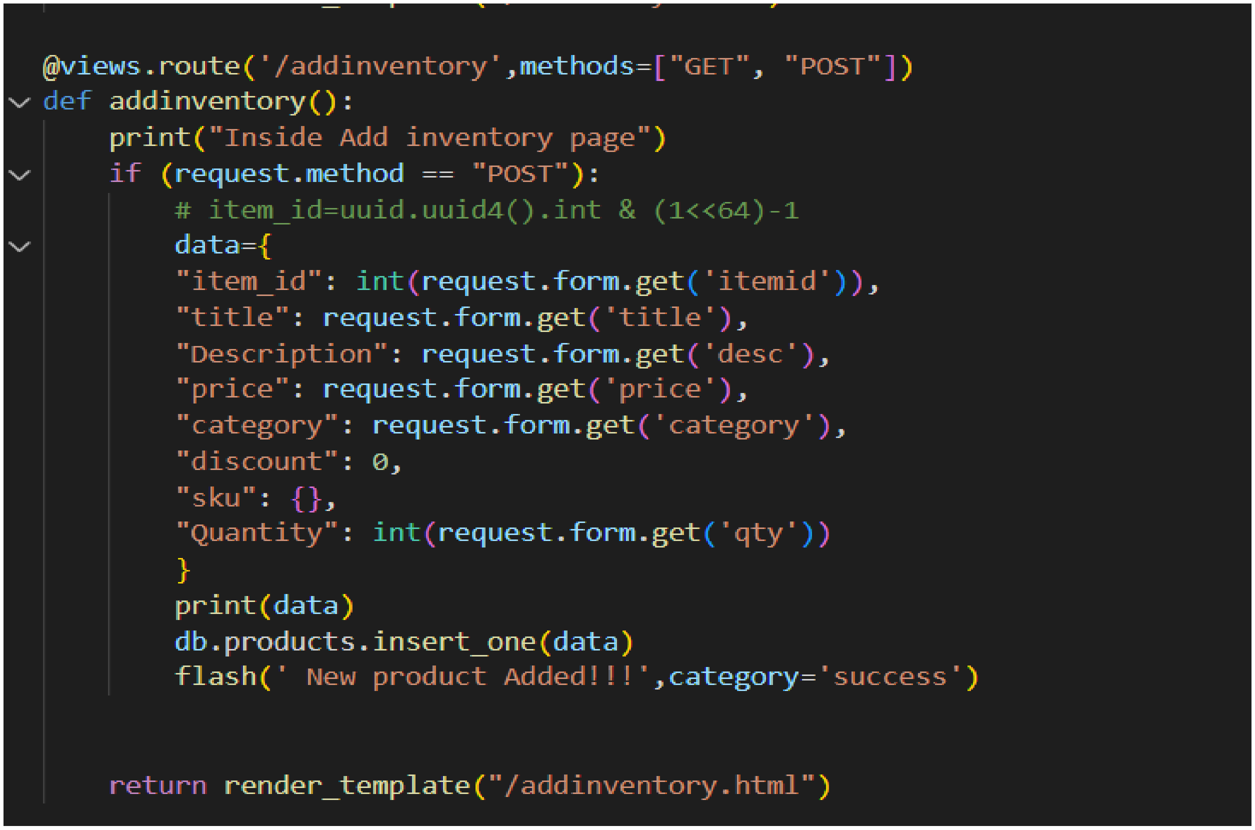
```
@views.route('/returns',methods=["GET", "POST"])
def returns():
    print("Inside Returns Function")
    temp=request.args
    temp=temp.to_dict(flat=False)
    returnid=temp["returnid"][0]
    li=returnid.split(',')
    orderid= li[0]
    itemid=li[1]
    item= db.orders.find_one({'order_id':orderid})
    ordered_time = item["time_order"]
    items= item["item_details"]
    temp={}
    for i in items:
        print(i['item_id'])
        if str(i['item_id'])==str(itemid):
            temp['qty']=i['qty']
            temp['discount']=i['discount']
            temp['price']=i['price']
            temp['category']=i['category']
    if(temp["category"] in ["vegetables","diary"]):
        current_time = datetime.datetime.now()
        duration = current_time - ordered_time
        duration_in_s = duration.total_seconds()
        days  = duration.days                          # Build-in datetime function
        days  = divmod(duration_in_s, 86400)[0]
        if (days>=1):
            flash(' This Item Cannot be returned',category='error')
            return redirect(url_for('views.Orders'))
    current_time = datetime.datetime.now()
    product = db.products.find_one({'item_id':int(itemid)})
    title = product['title']
    data={ "return_id":returnid, ···
    db.returns.insert_one(data)
    flash(' Order Returned initiated',category='success')
    return redirect(url_for('views.home'))
```

Inventory (Veiw & Update) code:-



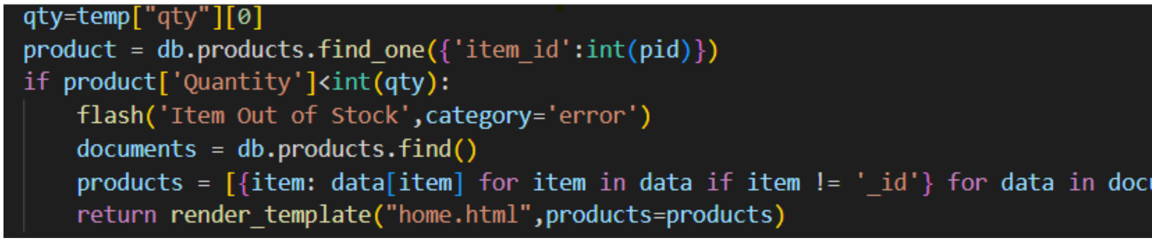
```
@views.route('/inventory',methods=["GET", "POST"])
def allinventory():
    print("Inside Admin inventory page ")
    documents = db.products.find()
    products = [{item: data[item] for item in data if item != '_id'} for data in documents]
    # orders=inventoryfun()
    return render_template("inventory.html",data=products)


@views.route('/inventoryupdate',methods=["GET", "POST"])
def inventoryupdate():

    if(request.method=="POST"):
        print("Inside Update Inventory method")
        db.products.update_one({
            "item_id" :int(request.form.get('itemid'))},
            {"$set":{
                "Quantity":int(request.form.get('qty')),
                "title":request.form.get('title'),
                "Description":request.form.get('desc'),
                "price":request.form.get('price'),
                "category":request.form.get('category')
                }
            })
        flash(' Item Details Updated!!',category='success')
        return redirect(url_for('views.allinventory'))
    return render_template("/inventory.html")
```

Inventory Add Code:-



```
@views.route('/addinventory',methods=["GET", "POST"])
def addinventory():
    print("Inside Add inventory page")
    if (request.method == "POST"):
        # item_id=uuid.uuid4().int & (1<<64)-1
        data={
        "item_id": int(request.form.get('itemid')),
        "title": request.form.get('title'),
        "Description": request.form.get('desc'),
        "price": request.form.get('price'),
        "category": request.form.get('category'),
        "discount": 0,
        "sku": {},
        "Quantity": int(request.form.get('qty'))
        }
        print(data)
        db.products.insert_one(data)
        flash(' New product Added!!!',category='success')


    return render_template("/addinventory.html")
```

Mongodb Queries :

- Login Query:-
  - user = db.Admin.find_one({"email":email})
- Register Query:
  - db.customers.insert_one(input_data)
- Home Page View Products:
  - documents = db.products.find()
- Queries in add cart
  - 

```
qty=temp["qty"][0]
product = db.products.find_one({'item_id':int(pid)})
if product['Quantity']<int(qty):
    flash('Item Out of Stock',category='error')
    documents = db.products.find()
    products = [{item: data[item] for item in data if item != '_id'} for data in docu
    return render_template("home.html",products=products)
```

- Payment Queries:-
  - user = db.customers.find_one({"email":email_id})
  - db.payment.insert_one(input_data)
- View Orders Query:-
  - documents = db.orders.find({'email_id':session["name"]})
  - single order = db.orders.find_one({'order_id':orderid})
- Returns Query:-
  - db.returns.update_one(
  - {"return_id" : returnid},{"$set":{"return_status":"Accepted"}})
  - db.products.update_one({'item_id':int(itemid)}, { "$set": { "Quantity": int(new_quantity) } })