



# **Distributed Systems**

**SE3020**

**Batch ID – Y3.S1.SE.WD.01**

**Group ID - -04-----**

*Express herb ayurvedic shopping platform*

## **Project Report**

## Group Details

	Surname with initials	Registration Number	Email
1	Jinasena H.D.S.S	IT21042560	it21042560@my.sliit.lk
2	Sandaruwan W.S.R	IT21041716	it21041716@my.sliit.lk
3	Lalanga S.P.H	IT21049590	It21049590@my.sliit.lk
4	Jayasinghe J.M.Y	IT21045158	it21045158@my.sliit.lk

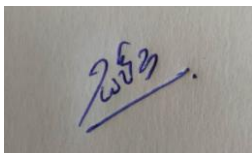
### Project Declaration

We, the members of CSSE\_21, hereby declare that our group project is entirely authentic and original. We have conducted thorough research and analysis to ensure that our work is not plagiarized or copied from any other sources.

We have followed all guidelines provided by our LIC and have complied with all ethical and academic standards.

We take full responsibility for the authenticity of our work and understand the implications of academic dishonesty. We have worked collaboratively to produce this project, and each member has contributed to the best of their abilities.

We hereby affirm that our project represents our honest effort and commitment to academic integrity, and we take pride in presenting it as our own.



Group Leader (Signature)  
Jinasena H.D.S.S

## Project Introduction

The online medical item buy and sell platform is a web application that allows medical item sellers to showcase their products and sell them to customers all over the world. This platform is designed to make the process of buying and selling medical items easier, faster, and more convenient.

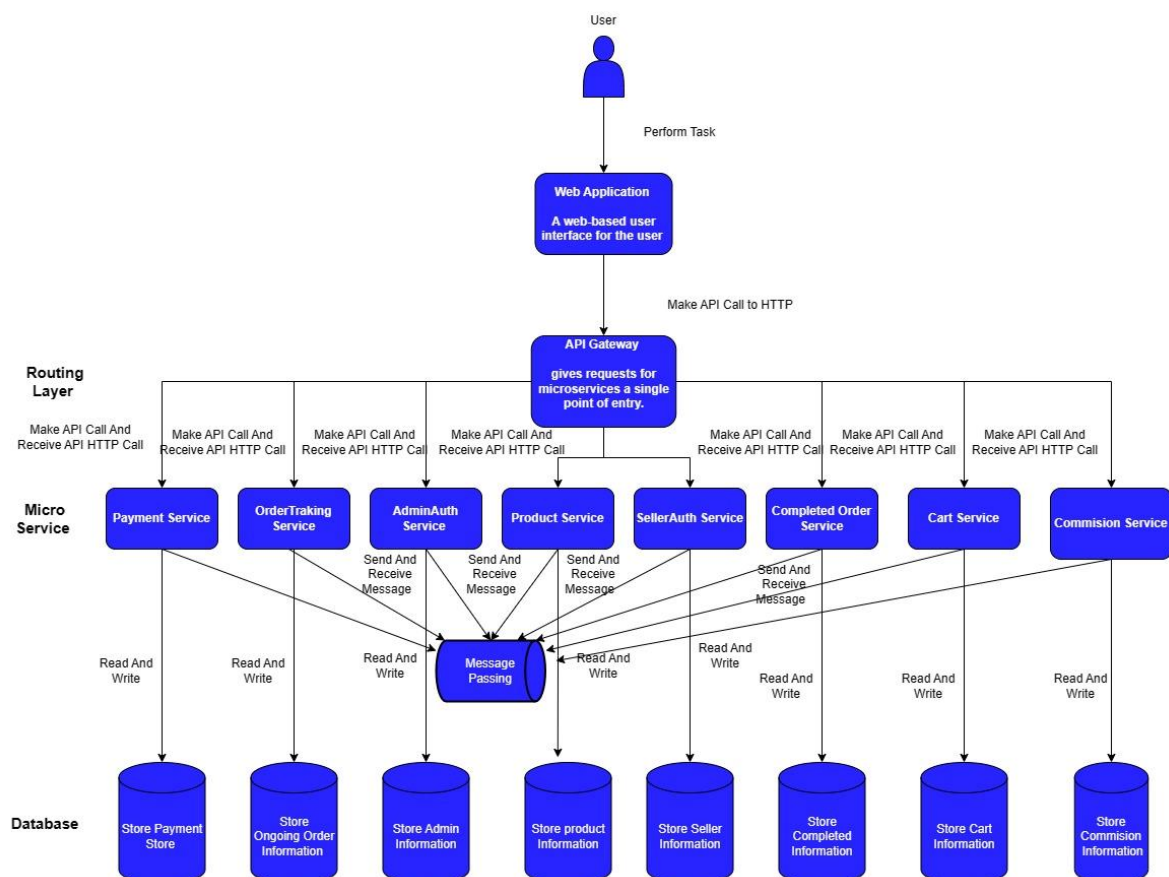
Through this website, medical item sellers are able to add their products to the online store, update previously added product details, and remove those products if they want. Customers can browse through a variety of medical products and easily find what they need. The shopping cart feature allows customers to add products to their cart and purchase them in a few easy steps.

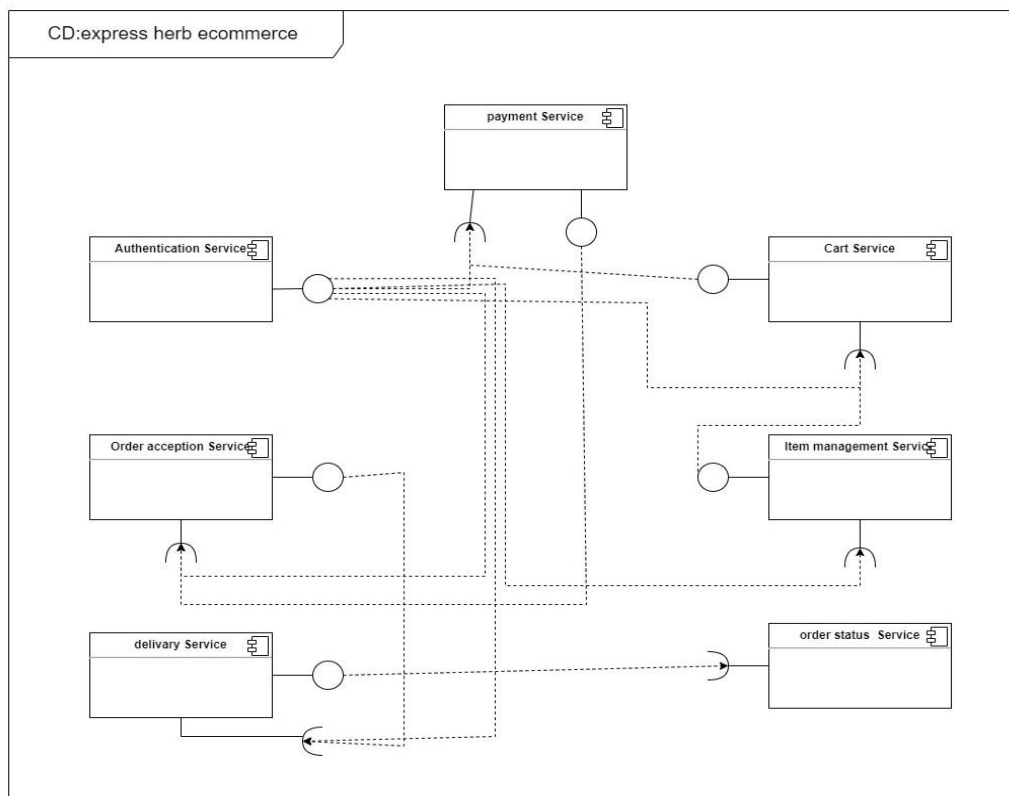
The customer dashboard is a feature that allows customers to track their incoming orders and get updates on the status of their shipments. The system also provides a secure and reliable card payment method for the process of paying bills, ensuring that customers can complete their transactions without any hassle.

To make things even more convenient for customers, the platform allows users to create their accounts, which can be used to save their details and make future purchases even faster. Similarly, sellers can also create their accounts, which enable them to manage their inventory, track their sales, and communicate with customers.

Overall, this online medical item buy and sell platform provides a seamless and user-friendly experience for both sellers and customers, making it easier for them to buy and sell medical items online.

## System Architecture





A modern application with services for products, carts, orders, sellers, and user authentication is meant to provide users with a seamless and useful user experience. It is developed on a microservice architecture. Each of the services is designed to handle a specific business domain or job and is built using a distinct database for increased scalability and dependability.

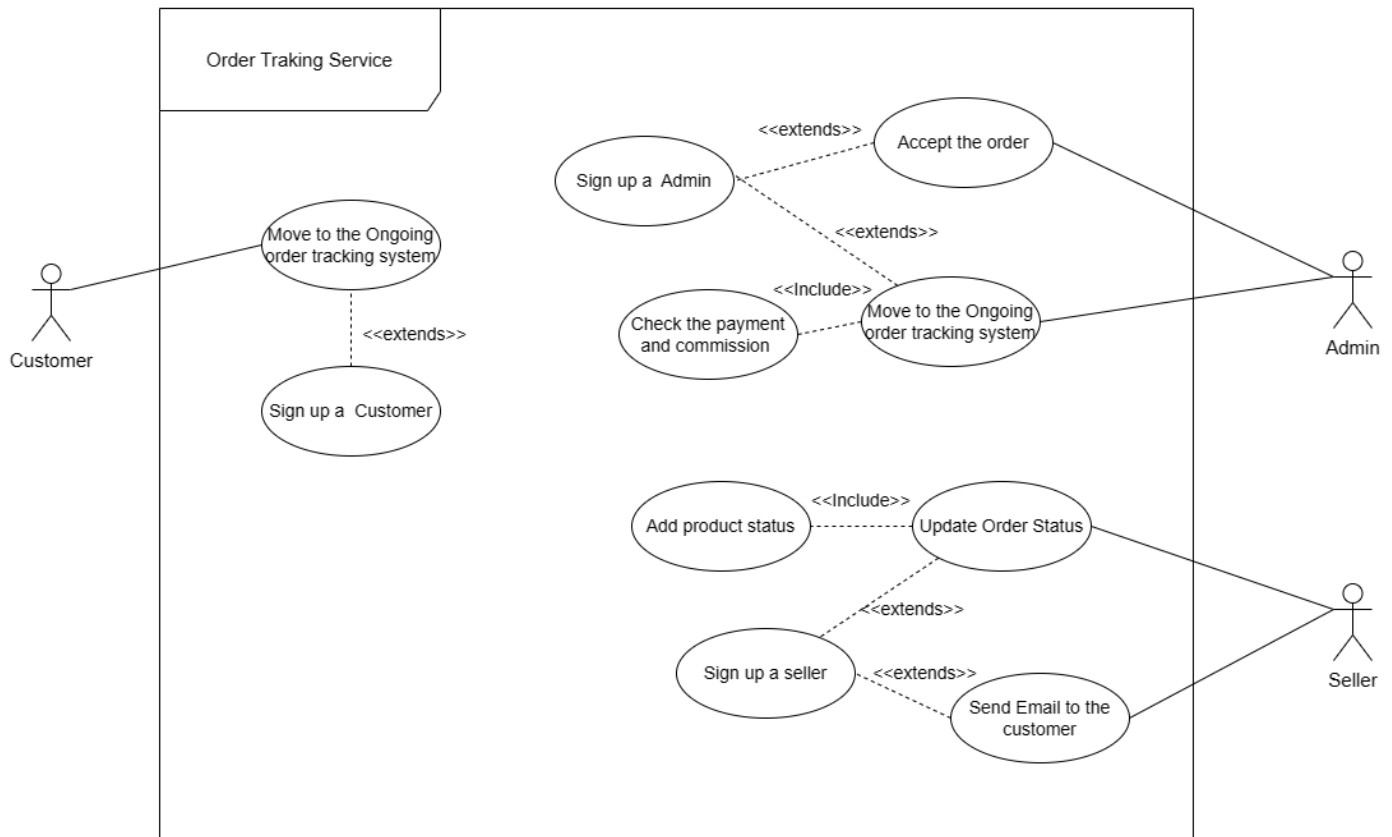
In charge of managing the product catalog and providing users with product information is the product service. It keeps track of information about the product's name, attributes, price, and availability. The Cart service controls the user's cart and also allows users to check out and add or remove items. The Order service is in charge of managing orders, including payment and shipping.

Asynchronous communication between the services can also be made possible by the app by employing message queues or event-driven structures. For instance, the Order service might broadcast an event once an order is placed, which the Seller service might subsequently use to update the product inventory.

Overall, this app's microservice architecture offers a number of advantages, such as increased scalability, robustness, and flexibility. Independent development and deployment of every service enables shorter development cycles and simpler maintenance.

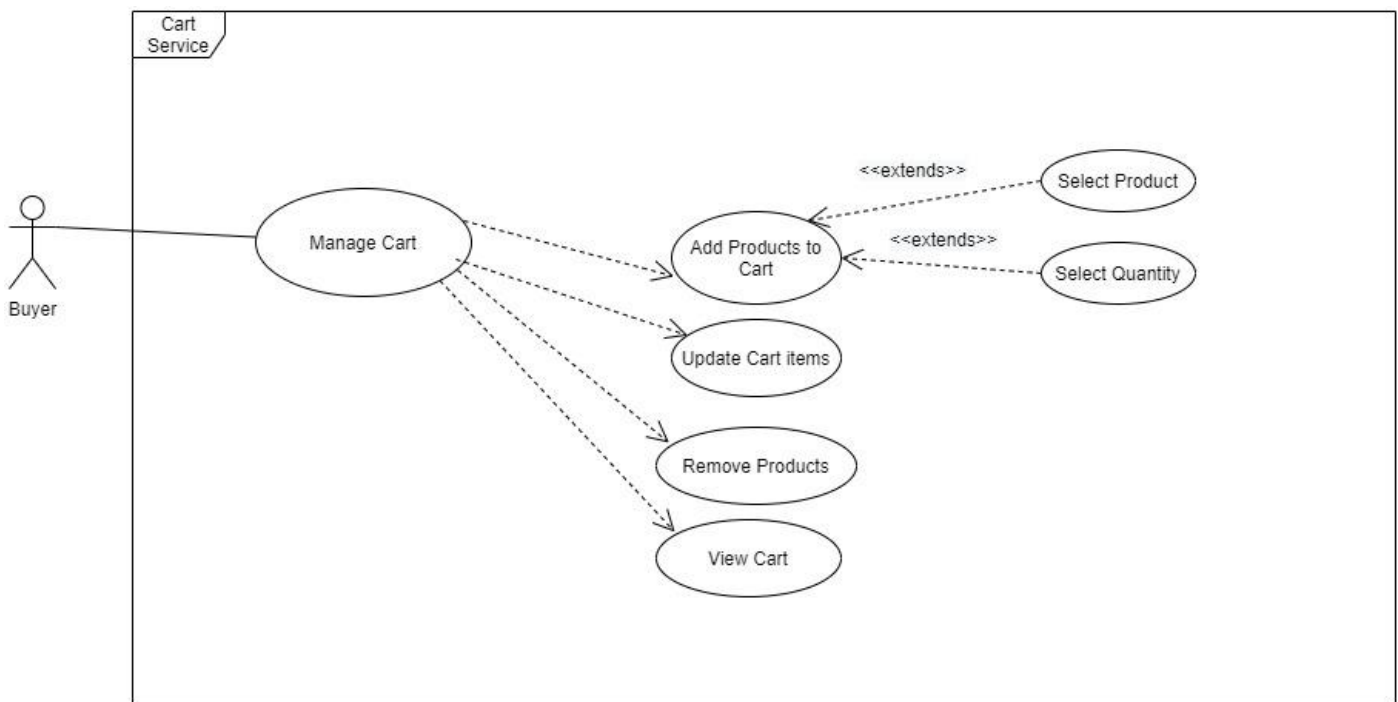
# Service Interfaces

## 1. Order tracking Management



The ordering item function allows the customers to browse and select products from a catalog or enter product information manually. Once the customer has selected their desired items, they proceed to the checkout process where they can choose from various payment and shipping options. Once the order is placed, the system generates a unique order ID that is used to track the order throughout the fulfillment process. The tracking order function allows customers to check the status of their orders in real time. The system provides updates on the order's progress, such as whether it is still processing, in transit, or has been delivered. The seller also updates the order status as it progresses through the fulfillment process, and this information is displayed to the customer in their user interface (UI). This allows the customer to stay informed and anticipate the arrival of their order. The complete order function is used to finalize a transaction after the order has been received. Once the order is complete, the seller can view the order details, such as the products ordered, shipping address, and payment information. This component of the complete order function allows the seller to view all the details related to a specific order, including the products ordered, payment information, and shipping address. This helps the seller ensure that the order was fulfilled correctly and that there were no errors. Overall, these functions work together to create a seamless ordering and fulfillment process for customers while providing sellers with the tools they need to manage orders and offer exceptional customer service. The tracking order function helps customers stay informed and anticipate the arrival of their orders, while the complete order function allows sellers to manage order details and offer discounts, fostering customer loyalty and driving repeat business.

## 2. Shopping-cart service



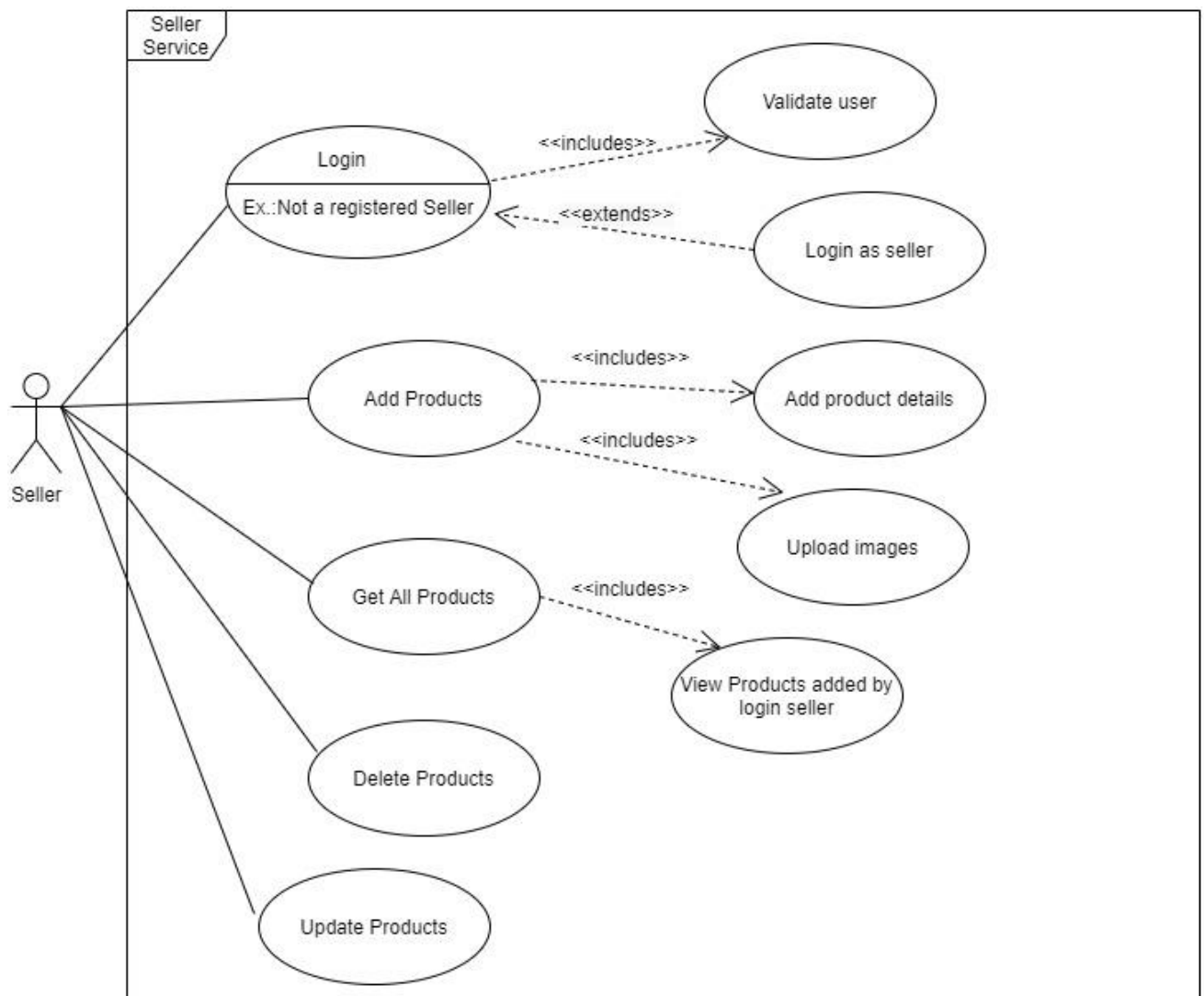
Any e-commerce platform must have a shopping cart service because it enables customers to manage their purchases more effectively by allowing them to add items to their basket, update their cart, delete items from their cart, and view their cart. An overview of the key features of the shopping cart service is provided below:

1. Adding items to the shopping cart: Customers can add items to their shopping carts by selecting the "Add to cart" button on the product page or the quick view modal. After then, the shopping cart service stores the product information, including the name, price, and quantity, in the basket of the buyer.

2. Updating the cart: Customers can change the number of goods in their cart or remove everything from it. When necessary, the shopping cart service updates the cart and updates the cart total.

3. Emptying the cart: By selecting the "Empty cart" button, customers can empty their carts. All items in the cart are removed by the shopping cart service.

### 3. Seller



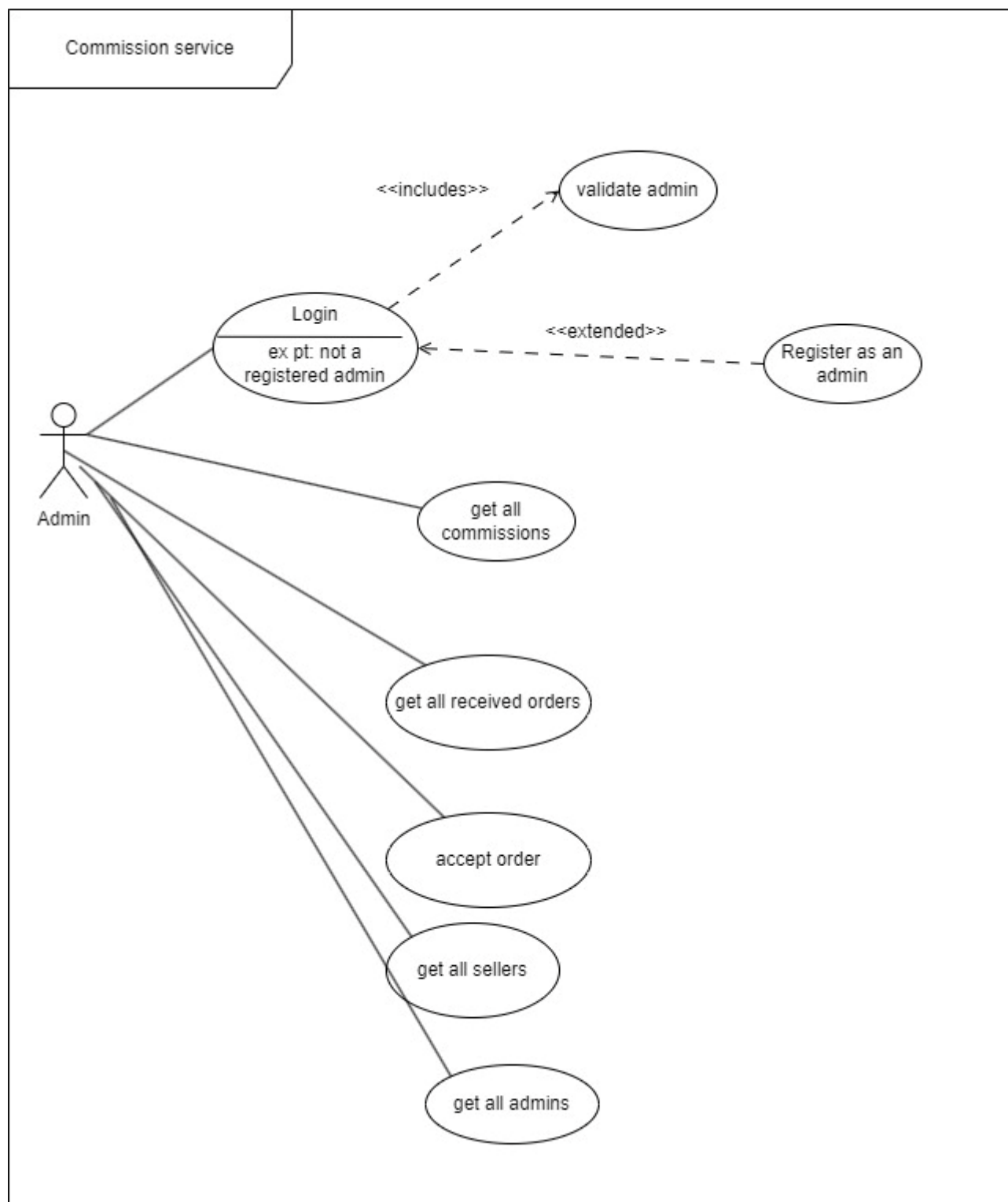
A platform for sellers to display their goods is what the seller service is intended to offer. To submit their products to the platform, sellers must first register on the website to do so. They are able to input product information, including prices, descriptions, and photographs. The seller dashboard allows them to view and manage their products after they have been added. The vendor can do a number of tasks from the dashboard, including editing or deleting the product information. Through the things Service, customers can view these things and buy them. When a customer makes a purchase, the Order Service will handle processing the order.

The workflow for the Seller Service is as follows:

- The vendor registers on the website and becomes a seller.
- The seller adds their products through the seller dashboard.
- The seller can see their products through the seller dashboard.
- The seller can update their products through the seller dashboard.
- The seller can delete their products through the seller dashboard.

The Seller can See their seller profile

## 4. Admin service





Managing and updating the website's content, including product accepting, price, and website design, is one of the main duties of the admin function. Additionally, the admin function might be in charge of controlling and keeping an eye on the operation of the website, user behavior, and sales statistics.

The admin role may also entail managing and keeping an eye on customer orders and accounts, including shipping, returns, and payment processing. Administrators may also be in charge of overseeing customer service functions like answering questions from customers, fixing problems, and handling complaints.

Admin server

```
import mongoose from 'mongoose'

import bcrypt from 'bcryptjs'
const Schema = mongoose.Schema;

const AdminSchema = new Schema({

  Admin_ID:{
    type:String,
    required: true
  },
  Full_Name:{
    type: String,
    required: true
  },
  Admin_Email:{
    type:String,
    lowercase:true,
    required: true
  },
  Job_title:{
    type:String,
    required: true
  },
  Contact_no:{
    type:String,
    required: true
  },
  Hash_password:{
    type:String,
    required: true
  },
  ProfilePicture:{
    type:String
  }
},{timestamps:true})

AdminSchema.virtual('password').set(function>Password){
  this.Hash_password =bcrypt.hashSync>Password, 8
});

AdminSchema.methods ={
  authenticate: function(){
    return bcrypt.compareSync>Password , this.Hash_password);
  }
}
```

```
export default mongoose.model("Admin",AdminSchema);
```

## Admin controller

```
import admin from '../models/AdminModels.js'

import bcrypt from 'bcryptjs'
import jwt from 'jsonwebtoken'

let refreshtokens = [];

export const AdminRegister = async (req, res) => {
  try{
    let file = 'N/A'
    if (req.file) {
      file = req.file.filename
    }
    const ExsistAdmin = await admin.findOne({ Admin_Email: req.body.Admin_Email });
    if (ExsistAdmin) {

      res.status(404).json({
        message: "Admin Already registered..!",
      })
    } else if (!ExsistAdmin) {
      const prefix = 'ADM'
      const Admin_ID = (prefix + Date.now())

      const Hash_password = await bcrypt.hash(req.body.Password, 10);
      const newAdmin = new admin({
        Admin_ID: Admin_ID,
        Full_Name: req.body.Full_Name,
        Admin_Email: req.body.Admin_Email,
        Job_title: req.body.Job_title,
        Contact_no: req.body.Contact_no,
        Hash_password: Hash_password,
        ProfilePicture: file

      });

      const newAcct = await newAdmin.save()
      if (newAcct) {

        res.status(201).json({
          message: "Registration Sucessfull..!",
          payload: newAcct
        })
      } else {
```

```

        res.status(400).json({
            message: "Somthing Went Wrong In Account Creating..!"
        })
    }

}
} catch (error) {
    res.status(500).json({
        message: "Somthing Went Wrong..!",
        error: error
    })
}

}
}

export const Signin = async (req, res) => {
    try {
        console.log(req.body.Admin_Email)
        const RegisteredAdmin = await admin.findOne({ Admin_Email: req.body.Admin_Email });
        // console.log(RegisteredAdmin)
        if (RegisteredAdmin) {
            const enterdPwd = req.body.Password;
            const dbPwd = RegisteredAdmin.Hash_password;
            // console.log(enterdPwd, dbPwd)
            const checkPwd = await bcrypt.compare(enterdPwd, dbPwd);
            // console.log(checkPwd)
            if (checkPwd) {
                const token = jwt.sign({ Admin_Email: req.body.Admin_Email },
process.env.JWT_TOKEN_KEY, { expiresIn: '1h' });
                const refreshToken = jwt.sign({ Admin_Email: req.body.Admin_Email },
process.env.REFRESH_TOKEN_KEY, { expiresIn: '5h' });
                // console.log("token "+token)
                // console.log("refresh token "+refreshToken)
                refreshtokens.push(refreshToken);
                res.status(201).json({
                    mesage: "Login Successfull..!",
                    token,
                    refreshToken,
                    payload:{RegisteredAdmin}
                })
            } else {
                res.status(401).json({
                    message: "Incorrect Password..!"
                })
            }
        } else {
            res.status(404).json({
                message: "Admin Not Registered..!"
            })
        }
    }
}

```

```

    })
  }

} catch (error) {
  console.log(error)
  res.status(500).json({
    message: "Server error..!",
    error: error
  })
}
}

export const tokenRefresh = (req, res, next) => {
  const refreshToken = req.body.refreshToken;
  if (refreshToken == null) {
    res.status(401).json({
      message: "Unauthorized..!"
    })
  } else if (!refreshtokens.includes(refreshToken)) {
    res.status(403).json({
      message: "Forbidden..!"
    })
  } else {
    jwt.verify(refreshToken, process.env.REFRESH_TOKEN_KEY, (err, user) => {
      if (err) {
        res.status(403).json({
          message: "Forbidden..!"
        })
      } else {
        const token = jwt.sign({ Admin_Email: req.body.Admin_Email },
process.env.JWT_TOKEN_KEY, { expiresIn: "1h" });
        res.status(201).json({
          message: "Session Extended..!",
          token
        })
      }
    })
  }
}

export const Signout = (req, res) => {
  try {
    const refreshToken = req.body.refreshToken;
    refreshtokens = refreshtokens.filter(token => token !== refreshToken);
    res.status(200).json({
      message: "Signout successful!",
    });
  } catch (error) {
    res.status(500).json({
      message: "Something went wrong!",
      error: error,
    });
  }
}

```

```

    });
  }
}

export const getAllAdmins = async (req, res) => {
  try {
    const allAdmins = await admin.find();
    if(allAdmins){
      res.status(200).json({
        message:"Fetched sucessfull..!",
        payload:allAdmins
      })
    }
  }catch(error){
    console.log(error)
  }
}

export const deleteAdmin = async(req,res) => {
  try{

    let userId = req.body.Admin_ID
    const success = await admin.findOneAndDelete({Admin_ID : userId})
    if(success){
      res.status(200).json({
        message:"deleted..!"
      })
    }
    else{
      res.status(400).json({
        message:"error..!"
      })
    }
  }catch{
    res.status(500).json({
      message:"server Error..!"
    })
  }
}

```

## Admin routes

```

import express from 'express';
import {AdminRegister, Signin, Signout, tokenRefresh, getAllAdmins ,deleteAdmin} from
'../controllers/AdminController.js'
import multer from 'multer';
// import requireSignIn from '../middleware/auth.js';

const router = express.Router();

```

```

const storage = multer.diskStorage({
  destination: function (req,file,cb){
    cb(null, 'uploadImages')
  },
  filename: function(req, file,cb){
    cb(null,Date.now() + '_' + file.originalname);
  }
})

const upload = multer({storage});

router.post('/Signup',upload.single('ProfilePicture'),AdminRegister);
router.post('/Signin',Signin);
router.delete('/Signout',Signout);
router.post('/Token',tokenRefresh);
router.get('/admins', getAllAdmins);
router.post('/deleteAdmin', deleteAdmin);

export default router

```

## admin server

```

import express from 'express'
import mongoose from 'mongoose'
import bodyParser from "body-parser"
import cors from 'cors'
import dotenv from 'dotenv'
import path from "path"
import { fileURLToPath } from "url"

const filePath = fileURLToPath(import.meta.url);
const dirName = path.dirname(filePath);

const app = express();
const PORT = process.env.PORT || 8041
dotenv.config()
app.use(cors())
app.use(bodyParser.json())
app.use(express.static(path.join(dirName, "uploadImages")));

const URL = process.env.MONGODB_URL;

mongoose.connect(URL,{
  useNewUrlParser: true,

```

```

    useUnifiedTopology: true,
  })

  // app.get("/", function(req,res){
  //     res.send("helloo docker...")
  // })

  app.listen(PORT, () =>{
    console.log("*****")
    console.log(`Admin Server Running on port : ${PORT}`)
  })

  const connection = mongoose.connection;
  connection.once("open", ()=>{
    console.log("MongoDB Connection success!")
    console.log("*****")
  })

  import admin from './routes/AdminRoutes.js'
  app.use("/Admin", admin);

```

docker file

```

FROM node:latest

WORKDIR /app

COPY . .

RUN apt-get update && \
    apt-get install -y build-essential && \
    npm install

EXPOSE 8041

ENTRYPOINT ["node", "server.js"]

```

Commission model

```

import mongoose from "mongoose";
const Schema = mongoose.Schema;

const ComSchema = new Schema({

  Com_ID: {
    type:String,
    require:true
  },
  Order_ID: {
    type: String,

```



```

        required: true
    },
    Total_Amount: {
        type: String,
        required: true,
    },
    Commission: {
        type: String,
        required: true,
    },
    },
    })
export default mongoose.model("Commission",ComSchema )

```

## commission controller

```

import commis from './comModel.js'

export const addCommission = async (req, res) => {
    console.log(req.body)
    try {
        const Order_ID = req.body.Order_ID
        const Total_Amount = req.body.Total_Amount

        const Commission = Total_Amount * 8 / 100;
        const prefix = 'COM'
        const Com_ID = (prefix + Date.now())

        const newcommission = new commis({
            Com_ID,
            Order_ID,
            Total_Amount,
            Commission
        })

        const response = await newcommission.save()
        if (response) {
            res.status(201).json({
                message: "Commission calculated...!",
                payload: newcommission
            })
        }
        else {
            res.status(404).json({
                message: 'error...!'
            })
        }
    } catch (error) {
        res.status(500).json({

```

```

        message: "Server error..!"
    })
}
}

export const getAll = async (req, res) => {

    const commissions = await commis.find()
    if(commissions){
        res.status(201).json({
            message: "Success..!!",
            payload: commissions
        })
    }else{
        res.status(400).json({
            message: "Error...!"
        })
    }
}
}

```

## Commission route

```

import express from "express";
const router = express.Router();
import {addCommission, getAll} from './comController.js'

router.post('/add', addCommission);
router.get('/get', getAll);

export default router

```

## commission docker

```

FROM node:latest

WORKDIR /app

COPY . .
RUN npm install

EXPOSE 8044

ENTRYPOINT ["node", "server.js"]

```

IT21042560 H.D.S.S jinasena

Order tracking service

Model and controller class

[11:33 PM] Jinasena H.D.S.S it21042560

```
const router = require("express").Router();

const nodemailer = require('nodemailer');

const {EMAIL, PASSWORD} = require('./env.js');

const Mailge = require('mailgen');

let OrderedItem = require("../modules/orderedItems.js");

const Mailgen = require("mailgen");
```

```
exports.AddItem = (async (req,res) => {

  //add router

  const order_id = req.body.order_id;

  const customer_name = req.body.customer_name;

  const address = req.body.address;

  const email = req.body.email;

  const status = req.body.status;

  const date = req.body.date;

  const newItem = new OrderedItem({

    order_id,

    customer_name,

    address,

    email,

    status,
```

```
    date

  })

  await newItem.save().then(()=>{

    res.json(`${order_id} added to the order tracking system `)

  }).catch((err) => {

    console.log(err);

  })

})
```

```
exports.UpdateItem = (async(req,res)=>{

  //update router

  let userId = req.params.id;

  const order_id = req.body.order_id;

  const customer_name = req.body.customer_name;

  const address = req.body.address;

  const email = req.body.email;

  const status = req.body.status;

  const date = req.body.date;

  const updateItem = {

    order_id,

    customer_name,

    address,
```

```
    email,  
  
    status,  
  
    date  
  }  
  
  //email seinding  
  
  let config = {  
  
    service:'gmail',  
  
    auth:{  
  
      user:EMAIL,  
  
      pass:PASSWORD  
    }  
  }  
}
```

```
let transporter = nodemailer.createTransport(config);
```

```
//set Header
```

```
let MailGenerator = new Mailgen({  
  
  theme:'default',  
  
  product: {  
  
    name: "Express Herb",  
  
    link: "https://mailgen.js/"  
  }  
})
```

```
let response = {
```

```
body:{  
    name:customer_name,  
    intro:`Your Order has `+ status + " !",  
    table:{  
        data:[  
            {  
                order:order_id,  
                address:address,  
                date:new Date()  
            }  
        ]  
    },  
    outro:"Hope to do more business !!!"  
}  
}
```

```
let mail = MailGenereto.generate(response)
```

```
let message = {  
    from : EMAIL,  
    to : email,  
    subject:"Place Order",  
    html: mail  
}
```

```
transpoter.sendMail(message)
```

```

const update = await OrderedItem.findByIdAndUpdate(userId, updateItem).then(() => {

  res.status(200).send({status: "Item Updated"})

}).catch((err) =>{

  console.log(err);

  res.status(500).send({status: "Error with updation data"});

})

})

```

```

exports.DeleteItem = (async (req,res) =>{

  //delete item

  let userId = req.params.id;

  await OrderedItem.findByIdAndDelete(userId).then(() => {

    res.status(200).send({status: "Order Successfully delivered!"});

  }).catch((err)=>{

    console.log(err.message);

    res.status(500).send({status: "Error with remove attendance", error: err.message});

  })

})

```

```

exports.GetOneItem = ( async (req,res) => {

  //get One Item

  let userId = req.params.id;

```

```
const odr = await OrderedItem.findById(userId)

.then((order) => {

  res.json(order);

})

.catch((err) => {

  res.status(500).send({status: "Error with finding data", error: err.message});

});

})
```

```
exports.GetAllItem = ((req,res) =>{

  //get All Item

  OrderedItem.find().then((order)=>{

    res.json(order)

  }).catch((err) =>{

    console.log(err)

  })

})
```

```
module.exports = router;
```

[11:34 PM] Jinasena H.D.S.S it21042560

```
const mongoose = require('mongoose');
```

```
const Schema = mongoose.Schema;
```



```
const orderItem = new Schema({
```

```
  order_id : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
  customer_name : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
  address : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
  email : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
  status : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
    date : {  
      type: Date,  
      required: true  
    }  
  
  })
```

```
const orderedItem = mongoose.model("orderedItems",orderedItem);
```

```
module.exports = orderedItem;
```

[11:34 PM] Jinasena H.D.S.S it21042560

```
const mongoose = require('mongoose');
```

```
const Schema = mongoose.Schema;
```

```
const orderItem = new Schema({
```

```
  order_id : {  
    type: String,  
    required: true  
  },
```

```
  customer_name : {  
    type: String,  
    required: true  
  },
```

```
    address : {  
      type: String,  
      required: true  
    },  
  
    email : {  
      type: String,  
      required: true  
    },  
  
    status : {  
      type: String,  
      required: true  
    },  
  
    date : {  
      type: Date,  
      required: true  
    }  
  })  
  
  const orderedI = mongoose.model("orderedItems",orderItem);  
  
  module.exports = orderedI;  
  
  const express = require('express');
```

```
const { AddItem,UpdateItem,DeleteItem,GetOneItem,GetAllItem } = require('../controller/orderedItems');

const router = express.Router();

router.post('/add', AddItem);

router.delete('/delete/:id',DeleteItem);

router.put('/update/:id',UpdateItem);

router.get("/get", GetAllItem);

router.get("/get/:id", GetOneItem);


module.exports = router;
```

FROM node:16

### **Docker file**

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 8040

CMD ["node", "server.js"]

## Complete order service

### Model class

```
const mongoose = require('mongoose');
```

```
const Schema = mongoose.Schema;
```

```
const CompletedOrder = new Schema({
```

```
  order_id : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
  customer_name : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
  address : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
  email : {
```

```
    type: String,
```

```
    required: true
```

```
  },
```

```
status : {  
  type: String,  
  required: true  
},  
  
date : {  
  type: Date,  
  required: true  
}  
  
})  
  
const order = mongoose.model("completedItems",CompletedOrder);  
module.exports = order;  
  
router  
  
const router = require("express").Router();  
  
let completedItem = require("../moduls/completedOrder");  
  
router.route("/add").post((req,res) => {  
  
  const order_id = req.body.order_id;  
  
  const customer_name = req.body.customer_name;  
  
  const address = req.body.address;
```

```
const email = req.body.email;
```

```
const status = req.body.status;
```

```
const date = req.body.date;
```

```
const newItem = new completedItem({
```

```
  order_id,
```

```
  customer_name,
```

```
  address,
```

```
  email,
```

```
  status,
```

```
  date
```

```
})
```

```
newItem.save().then(()=>{
```

```
  res.json(`${order_id} added to the order tracking system `)
```

```
}).catch((err) => {
```

```
  console.log(err);
```

```
})
```

```
})
```

```
router.route("/delete/:id").delete(async (req,res) =>{
```

```
  let userId = req.params.id;
```

```
  await completedItem.findByIdAndDelete(userId).then(() => {
```

```
    res.status(200).send({ status: "Order Successfully delivered!" });

  }).catch((err)=>{

    console.log(err.message);

    res.status(500).send({ status: "Error with remove attendance", error: err.message });

  })

})

router.route("/get").get((req,res) =>{

  completedItem.find().then((order)=>{

    res.json(order)

  }).catch((err) =>{

    console.log(err)

  })

})

module.exports = router;
```

## **docker**

FROM node:16

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

COPY . .



EXPOSE 8028

CMD ["node", "server.js"]

### **Server class**

```
const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require("body-parser");

const cors = require("cors");

const dotenv = require("dotenv");

const app = express();

require("dotenv").config();

const PORT = process.env.PORT || 8022

app.use(cors());

app.use(bodyParser.json());

const URL = process.env.MONGODB_URL;

mongoose.connect(URL,{

})

const connection = mongoose.connection;
```

```
connection.once("open", ()=>{

  console.log("MongoDB Connection success!")

})

app.listen(PORT, () =>{

  console.log(`Server is up and running on port ${PORT}`)

})

const completedOrderItem = require("./routes/completedOrder.js");

app.use("/completedOrder", completedOrderItem);
```

**IT21049590 lalanga S.P.H**

**Product service**

**Model class**

```
import mongoose from "mongoose";

const productSchema = new mongoose.Schema({

  name: {

    type: String,

    required: true,

  },

  price: {

    type: Number,

    required: true,

  },

  category: {
```

```
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  quantity: {
    type: Number,
    required: true,
  },
  image: {
    type: String,
  },
  Email: {
    type: String,

  },
  SellerId:{
    type:String,
  },

});

export default mongoose.model("Product", productSchema);
```

## **controller class**

```
import express from "express";

const router = express.Router();

import Product from "../models/Product.js";

import multer from "multer";

import fs from "fs";

import path from "path";

var app = express();

import asyncHandler from "express-async-handler";

export const addProduct = async (req, res) => {

  let userId1 = req.params.id;

  let file = "N/A";

  if (req.file) {

    file = req.file.filename;

  }

  const name = req.body.name;

  const price = req.body.price;

  const category = req.body.category;

  const description = req.body.description;

  const quantity = req.body.quantity;

  const SellerId = userId1;

  console.log(req.body);
```

```
const newProduct = new Product({

  name,

  price,

  category,

  description,

  quantity,

  SellerId,

  image: file,

});

console.log(newProduct);

newProduct

  .save()

  .then(() => {

    res.json("Product Added");

  })

  .catch((err) => {

    console.log(err);

  });

};

export const viewProducts = async (req, res, next) => {

  let userId1 = req.params.id;

  await Product.find({ SellerId: userId1 })

    .then((Product) => {

      res.json(Product);

    })

  .catch((err) => {

    console.log(err);

  });

};
```

```

    //res.status(200).json(response);

  })

  .catch((err) => {

    console.log(err);

  });

};

export const viewAll = async (req, res, next) => {

  await Product.find()

  .then((Product) => {

    res.json(Product);

    //res.status(200).json(response);

  })

  .catch((err) => {

    console.log(err);

  });

};

export const updateProduct = async (req, res) => {

  let userId = req.params.id;

  const { name, price, category, description, quantity } = req.body;

  const updateProduct = {

    name,

    price,

    category,

```

```
    description,

    quantity,

  };

  const update = await Product.findByIdAndUpdate(userId, updateProduct)

    .then(() => {

      res.status(200).send({ status: "Product updated " });

    })

    .catch((err) => {

      console.log(err);

      res

        .status(500)

        .send({ status: "Error with updating data", error: err.message });

    });

  // return res.status(201).json({ updateDrivers })

};
```

```
export const deleteProduct = async (req, res) => {

  let userId = req.params.id;

  await Product.findByIdAndDelete(userId)

    .then(() => {

      res.status(200).send({ status: "Product deleted" });

    })

    .catch((err) => {

      console.log(err.message);

      res

        .status(500)
```

```

        .send({ status: "Error with delete Product", error: err.message });

    });

};

export const getProductById = async (req, res) => {

    let id = req.params.id;

    await Product.findById(id)

        .then((response) => {

            res.status(200).json(response);

            console.log(res);

        })

        .catch((err) => {

            res

                .status(500)

                .send({ status: "Error with get product", error: err.message });

        });

};

```

## Router class

- import express from "express";

 const router = express.Router();

 import multer from "multer";

 import path from "path";

 import {

 addProduct,



```

    viewProducts,

    updateProduct,

    deleteProduct,

    getProductById,

    viewAll,
  } from "../controllers/Product-controller.js";

// import { uploadPhoto, productImgResize } from "../middleware/uploadImages.js";

//add new product

const storage = multer.diskStorage({

  destination: function (req, file, cb) {

    cb(null, "uploads");

  },

  filename: function (req, file, cb) {

    cb(null, Date.now() + "_" + file.originalname);

  },

});

const upload = multer({ storage });

router.post("/addProduct/:id", upload.single("image"), addProduct);

router.get("/viewProducts/:id", viewProducts);

router.put("/updateProduct/:id", updateProduct);

router.delete("/deleteProduct/:id", deleteProduct);

router.get("/getProductById/:id", getProductById);

router.get("/viewAll", viewAll);

export default router;

```

has context menu

## **server**

```
import express from "express";

import mongoose from "mongoose";

import bodyParser from "body-parser";

import cors from "cors";

import dotenv from "dotenv";

const app = express();

import path from "path";

import { fileURLToPath } from "url";

const filePath = fileURLToPath(import.meta.url);

const dirName = path.dirname(filePath);

dotenv.config();

const PORT = process.env.PORT || 8040;

app.use(express.static(path.join(dirName, "uploads")));

app.use(cors());

app.use(bodyParser.json());

const URL = process.env.MONGODB_URL;

mongoose.connect(URL, { });

const connection = mongoose.connection;
```

```
connection.once("open", () => {  
  console.log("MongoDB Connection success!");  
});  
  
app.listen(PORT, () => {  
  console.log(`Server is up and running on port ${PORT}`);  
});  
  
// const product = require("./routes/Product-routes");  
  
// app.use('/Product', product);  
  
import product from "./routes/Product-Route.js";  
app.use("/Product", product);
```

### **docker file**

FROM node:16

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 8040

```
CMD ["node", "server.js"]
```

## **Seller login service**

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

```
const Schema = mongoose.Schema;
```

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

```
const SellerSchema = new Schema(
```

```
{
```

```
  Seller_Id: {
```

```
    type: String,
```

```
    required: true,
```

```
  },
```

```
  FirstName: {
```

```
    type: String,
```

```
    required: true,
```

```
  },
```

```
  LastName: {
```

```
    type: String,
```

```
    required: true,
```

```
  },
```

```
  Email: {
```

```
    type: String,
```

```
    required: true,
```

```
  },
```

```
  Contact_no: {
```

```
    type: String,
    required: true,
  },
  Hash_password: {
    type: String,
    required: true,
  },
  ProfilePicture: {
    type: String,
  },
},
{ timestamps: true }
);
```

```
SellerSchema.virtual("password").set(function (Password) {
  this.Hash_password = bcrypt.hashSync(Password, 8);
});
```

```
SellerSchema.methods = {
  authenticate: function () {
    return bcrypt.compareSync(password, this.Hash_password);
  },
};
```

```
module.exports = mongoose.model("Admin", SellerSchema);
```

## controller class

- const { response } = require("express");  
  
const Seller = require("../models/SellerModel");  
  
const bcrypt = require("bcrypt");  
  
const jwt = require("jsonwebtoken");  
  
let refreshtokens = [];  
  
var x;  
  
  
exports.SellerRegistration = async (req, res) => {  
  
 try {  
  
 let file = "N/A";  
  
 if (req.file) {  
  
 file = req.file.filename;  
  
 }  
  
 const ExsistSeller = await Seller.findOne({ Email: req.body.Email });  
  
 console.log(!ExsistSeller);  
  
 if (ExsistSeller) {  
  
 res.status(404).json({  
  
 message: "Seller Already registered..!",  
  
 });  
  
 } else if (!ExsistSeller) {  
  
 const prefix = "SID";  
  
 const S\_ID = prefix + Date.now();  
  
  
  
 const Hash\_password = await bcrypt.hash(req.body.Hash\_password, 10);  
  
 const newSeller = new Seller({  
  
 Seller\_Id: S\_ID,

```

        FirstName: req.body.FirstName,

        LastName: req.body.LastName,

        Email: req.body.Email,

        Contact_no: req.body.Contact_no,

        Hash_password: Hash_password,

        ProfilePicture: file,

    });

    const newAcct = await newSeller.save();

    if (newAcct) {

        res.status(201).json({

            message: "Registration Sucessfull..!",

            payload: newAcct,

        });

    } else {

        res.status(400).json({

            message: "Somthing Went Wrong In Account Creating..!",

        });

    }

}

} catch (error) {

    res.status(500).json({

        message: "Somthing Went Wrong..!",

        error: error,

    });

}

};

```

```

exports.Signin = async (req, res) => {
  try {
    const RegisterdSeller = await Seller.findOne({ Email: req.body.Email });
    console.log(RegisterdSeller);
    x = RegisterdSeller;
    if (RegisterdSeller) {
      const enterdPwd = req.body.Hash_password;
      const dbPwd = RegisterdSeller.Hash_password;
      // console.log(enterdPwd,dbPwd)
      const checkPwd = await bcrypt.compare(enterdPwd, dbPwd);
      console.log(checkPwd);
      if (checkPwd) {
        const token = jwt.sign(
          { Email: req.body.Email },
          process.env.JWT_TOKEN_KEY,
          { expiresIn: "1h" }
        );
        const refreshToken = jwt.sign(
          { Email: req.body.Email },
          process.env.REFRESH_TOKEN_KEY,
          { expiresIn: "24h" }
        );
        // console.log("token "+token)
        // console.log("refresh token "+refreshToken)
        refreshtokens.push(refreshToken);
        res.status(201).json({

```



```

        message: "Login Successfull..!",
        token,
        refreshToken,
        payload: { RegisterdSeller },
    });
} else {
    res.status(401).json({
        message: "Incorrect Password..!",
    });
}
} else {
    res.status(404).json({
        message: "Seller Not Registered..!",
    });
}
} catch (error) {
    console.log(error);
    res.status(500).json({
        message: "Server error..!",
        error: error,
    });
}
};

```

```

exports.tokenRefresh = (req, res, next) => {
    const refreshToken = req.body.refreshToken;
    if (refreshToken == null) {

```

```

    res.status(401).json({
      message: "Unauthorized..!",
    });
  } else if (!refreshTokens.includes(refreshToken)) {
    res.status(403).json({
      message: "Forbidden..!",
    });
  } else {
    jwt.verify(refreshToken, process.env.REFRESH_TOKEN_KEY, (err, user) => {
      if (err) {
        res.status(403).json({
          message: "Forbidden..!",
        });
      } else {
        const token = jwt.sign(
          { Admin_Email: req.body.Admin_Email },
          process.env.JWT_TOKEN_KEY,
          { expiresIn: "1h" }
        );
        res.status(201).json({
          message: "Session Extended..!",
          token,
        });
      }
    });
  }
};

```

```
exports.Signout = (req, res) => {  
  try {  
    const refreshToken = req.body.refreshToken;  
    refreshtokens = refreshtokens.filter((token) => token !== refreshToken);  
    res.status(200).json({  
      message: "Signout successful!",  
    });  
  } catch (error) {  
    res.status(500).json({  
      message: "Something went wrong!",  
      error: error,  
    });  
  }  
};
```

```
exports.getAllSeller = async (req, res) => {  
  try {  
    const allSeller = await Seller.find();  
    if (allSeller) {  
      res.status(200).json({  
        message: "Fetched Successfull..!",  
        payload: allSeller,  
      });  
    }  
  } catch (error) {  
    console.log(error);  
  }  
};
```

```

    }
};

exports.getOneSeller = async (req, res) => {

    let Seller_Id = req.params.Seller_Id;

    const RegisterdSeller = await Seller.findOne({

        Seller_Id: req.params.id,

    });

    res.status(201).json({

        // mesage: "Login Successfull..!",

        // payload: { RegisterdSeller },

        RegisterdSeller,

    });

};

exports.RegisterdSeller = x;

```

has context menu

## router class

- `const express = require("express");`
- `const {`
- `SellerRegistration,`
- `Signin,`
- `Signout,`
- `tokenRefresh,`
- `getAllSeller,`
- `getOneSeller,`

```
} = require("../controllers/SellerController");

//const requireSignin = require('../middleware/auth.js')

const multer = require("multer");

const path = require("path");

const router = express.Router();


const storage = multer.diskStorage({

  destination: function (req, file, cb) {

    cb(null, "UploadImage");

  },

  filename: function (req, file, cb) {

    cb(null, Date.now() + "_" + file.originalname);

  },

});

const upload = multer({ storage });


router.post("/Signup", upload.single("ProfilePicture"), SellerRegistration);

router.post("/Signin", Signin);

router.delete("/Signout", Signout);

router.post("/Token", tokenRefresh);

router.get("/seller", getAllSeller);

router.get("/seller/:id", getOneSeller);


module.exports = router;
```

has context menu

## **server**

[11:41 PM] Lalanga S.P H it21049590

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

const mongoose = require("mongoose");

const bodyParser = require("body-parser");

const cors = require("cors");

const dotenv = require("dotenv");

const app = express();

const seller = require("./routes/SellerRoute.js");

require("dotenv").config();

const PORT = process.env.PORT || 8050;

app.use(cors());

app.use(bodyParser.json());

const URL = process.env.MONGODB_URL;

mongoose.connect(URL, {});

const connection = mongoose.connection;

connection.once("open", () => {

  console.log("MongoDB Connection success!");

});
```

```
app.listen(PORT, () => {  
  console.log(`Server is up and running on port ${PORT}`);  
});  
app.use("/seller", seller);
```

**docker**

**FROM** node:16

**WORKDIR** /usr/src/app

**COPY** package\*.json ./

**RUN** npm install

**COPY** . .

**EXPOSE** 8050

**CMD** ["node", "server.js"]

IT21045158 J.m.y Jayasinghe

Cart server

controller class

```
import Cart from '../models/CartModels.js'
```

```
// import axios from 'axios'
```

```
import nodemailer from 'nodemailer'
```

```
import tls from 'tls'
```

```
import fs from 'fs'
```

```
export const Addcart = async (req, res) => {
```

```
  try {
```

```
    // <<<<<<< HEAD
```

```
      // const Order_ID = req.body.Order_ID
```

```
      const Product_ID = req.body.Product_ID;
```

```
    // =====
```

```
      const prefix = 'OID'
```

```
      const order_ID = (prefix + Date.now())
```

```
      const Order_ID = order_ID
```

```
    // >>>>>>> 3247cfb815ef05c6643d59a7aa13e4055fb70579
```

```
      const Seller_ID = req.body.Seller_ID
```

```
      const Customer_Name = req.body.Customer_Name
```

```
      const Address = req.body.Address
```

```
      const Phone_No = req.body.Phone_No
```

```
      const Email = req.body.Email
```

```
      const Total_Amount = req.body.Total_Amount
```



```
const Delivery = req.body.Delivery
```

```
const Zip = req.body.Zip
```

```
const State = req.body.State
```

```
const newCart = new Cart({
```

```
    Order_ID,
```

```
    Product_ID,
```

```
    Seller_ID,
```

```
    Customer_Name,
```

```
    Address,
```

```
    Phone_No,
```

```
    Email,
```

```
    Total_Amount,
```

```
    Delivery,
```

```
    Zip,
```

```
    State
```

```
})
```

```
const response = newCart.save()
```

```
if (response) {
```

```
    res.status(200).json({
```

```
        message: "success..!",
```

```
        payload: {
```

```
            cart: newCart
```

```
        }
```

```

        })
    }
    else{
        res.status(401).json({
            message: "cart error..!"
        })
    }

} catch (error) {
    res.status(500).json({
        message: "Server error..!"
    })
}

}

export const getCart = async (req, res) => {

    const allCart = await Cart.find().sort({ date: -1 });
    if (allCart) {
        res.status(200).json({
            message: "Fetched Successfully..!",
            payload: allCart
        })
    }
    else {

```

```
    res.status(404).json({
      message: "error fetched..!"
    })
  }
}

export const deleteCart = async (req, res) => {
  let Oid = req.body.oid

  try{

    const success = await Cart.findOneAndDelete({ Order_ID : Oid })

    if (success) {
      res.status(200).json({
        message: "Delete successfull..!"
      })

    } else {
      res.status(400).json({
        message: "Delete unsuccessful..!"
      })
    }
  } catch(error){
    res.status(500).json({
      message:"server error..!"
    })
  }
}
```

```
}

// this is for mails

export const deleteCartM = async (req, res) => {

  console.log(req.body)

  let Oid = req.body.oid

  const customer_name = req.body.Customer_Name

  const Email = req.body.email

  try{

    const success = await Cart.findOneAndDelete({ Order_ID : Oid })

    const transporter = nodemailer.createTransport({

      host: 'smtp.gmail.com',

      port: 587,

      secure: false,

      auth: {

        user: 'webinctechnology@gmail.com',

        pass: 'dvymzjhgvxgyhpzg'

      },

      tls: {

        rejectUnauthorized: false

      }

    });

    const mailOptions = {
```

from: process.env.EMAIL,

to: Email,

subject: 'Your Order has been rejected..!'

text: `Dear \${customer\_name},\n\nWe regret to inform you that we are unable to process your recent order(order id is \${Oid}) at this time due to some reason of our company. We apologize for any inconvenience this may have caused you.\n\nIf you have any questions or concerns, please contact our customer support team at [011-2456895].\n\nThank you for your understanding.\n\nBest regards,\nThe iHerb team`

};

transporter.sendMail(mailOptions, function(error, info) {

if (error) {

console.log(error);

} else {

console.log('Email sent: ' + info.response);

}

});

if (success) {

res.status(200).json({

message: "Delete successfull..!"

})

} else {

res.status(400).json({

message: "Delete unsuccessful..!"

})

}

}catch(error){

```
    res.status(500).json({  
      message:"server error..!"  
    })  
  }  
  
}
```

## Model class

```
import mongoose from 'mongoose'  
  
const Schema = mongoose.Schema;  
  
const CartSchema = new Schema({  
  
  Order_ID: {  
    type: String  
  },  
  Product_ID:{  
    type:String  
  },  
  Seller_ID: {  
    type: String  
  },  
  Customer_Name: {  
    type: String  
  },  
  Address: {  
    type: String
```

```
    },  
    Zip: {  
      type: String  
    },  
    State: {  
      type: String  
    },  
    Phone_No: {  
      type: String  
    },  
    Email: {  
      type: String,  
      lowercase: true  
    },  
    Total_Amount: {  
      type: String  
    },  
  
    Delivery: {  
      type: String  
    },  
  
  }, {timestamps: true})  
  
export default mongoose.model("Cart", CartSchema);  
.
```

## Router class

[11:43 PM] Jinasena H.D.S.S it21042560

```
import express from 'express';

const router = express.Router();

import {Addcart, getCart,deleteCart,deleteCartM} from '../controller/cartController.js'

router.post('/addcart',Addcart);

router.get('/getCart',getCart);

router.post('/deleteCart',deleteCart);

router.post('/deleteCartM',deleteCartM);

export default router
```

## docker file

```
FROM node:16

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 8040

CMD ["node", "server.js"]
```



