ABSTRACT

MedCart is a full-stack online pharmacy e-commerce platform designed to provide users with a seamless and secure experience for purchasing medicines and healthcare products. The system allows customers to browse categorized products, add items to their cart, choose preferred payment methods (Cash on Delivery or online payment), and track their orders. A responsive frontend ensures ease of use across devices, while backend APIs handle data management, authentication, and order processing.

The platform also features an admin dashboard for managing inventory, monitoring orders, and updating product details. Real-time interactions between users and the server are facilitated through secure RESTful APIs and MongoDB integration. MedCart aims to bridge the gap between healthcare access and convenience, offering users a reliable digital solution for medicine delivery from the comfort of their homes.

INTRODUCTION

MedCart is a full-stack online pharmacy e-commerce application built to simplify access to medicines and healthcare essentials. The platform allows users to create an account, browse categorized medical products, add items to their cart, and place orders using a secure checkout system. It supports multiple payment options, including Cash on Delivery and online payments. The application features a user-friendly frontend built with modern web technologies and a robust backend that handles authentication, product management, and order processing. Admins can manage product inventory, view orders, and update order statuses via a dedicated dashboard. MedCart aims to provide a convenient, reliable, and efficient online solution for users to access pharmacy services from their homes.

SCOPE

MedCart enables users to browse, search, and purchase pharmacy products online with secure payment and order tracking. It includes user registration, cart management, and a checkout process. Admins can manage products and monitor orders through a dashboard. The platform focuses on enhancing accessibility, user convenience, and efficient medicine delivery through a scalable e-commerce system.

SYSTEM FEATURES

- User registration and login
- Secure authentication
- Browse products by category
- Real-time product search
- Add/remove items to cart
- View cart summary
- Checkout with shipping details
- Cash on Delivery and online payment
- Order placement and confirmation
- View past orders
- Order status tracking
- Admin login access
- Add/edit/delete products (Admin)
- View and manage orders (Admin)
- Responsive UI for all devices

SYSTEM CONTEXT

MedCart is an online pharmacy system where users browse and purchase medicines, while admins manage products and orders. It interacts with users, admins, and payment gateways to process and deliver medical orders efficiently.

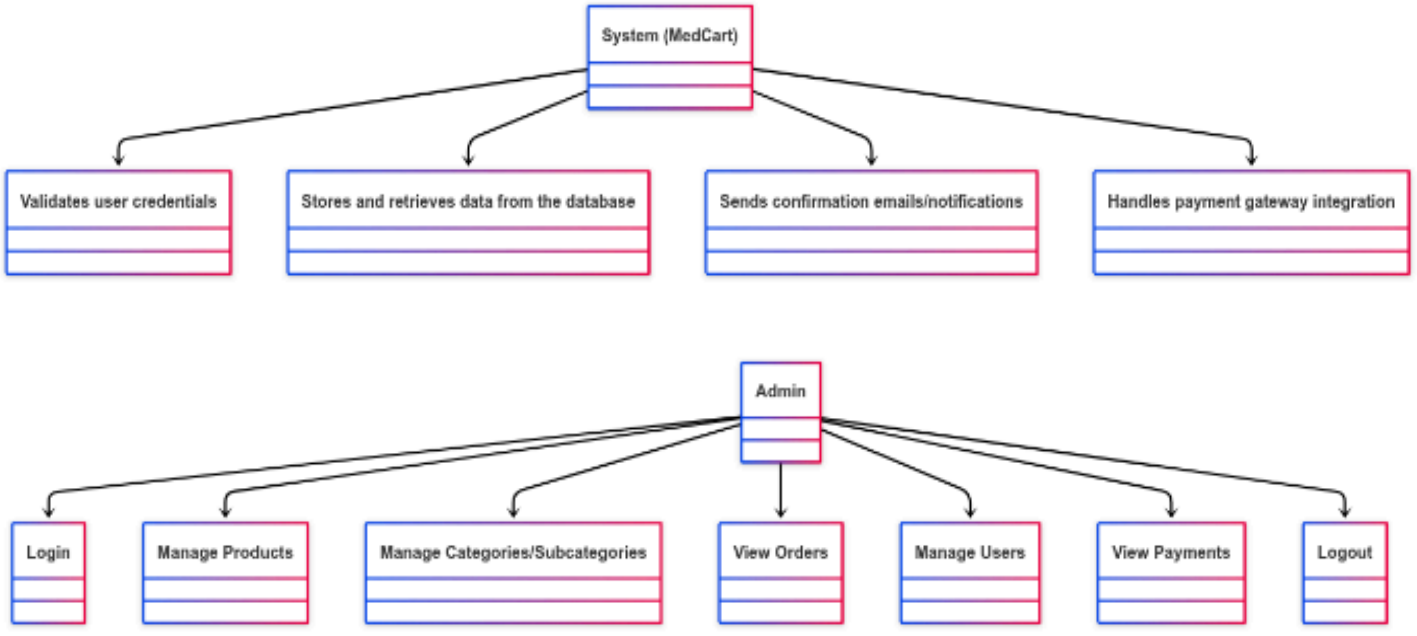
Functional Requirements

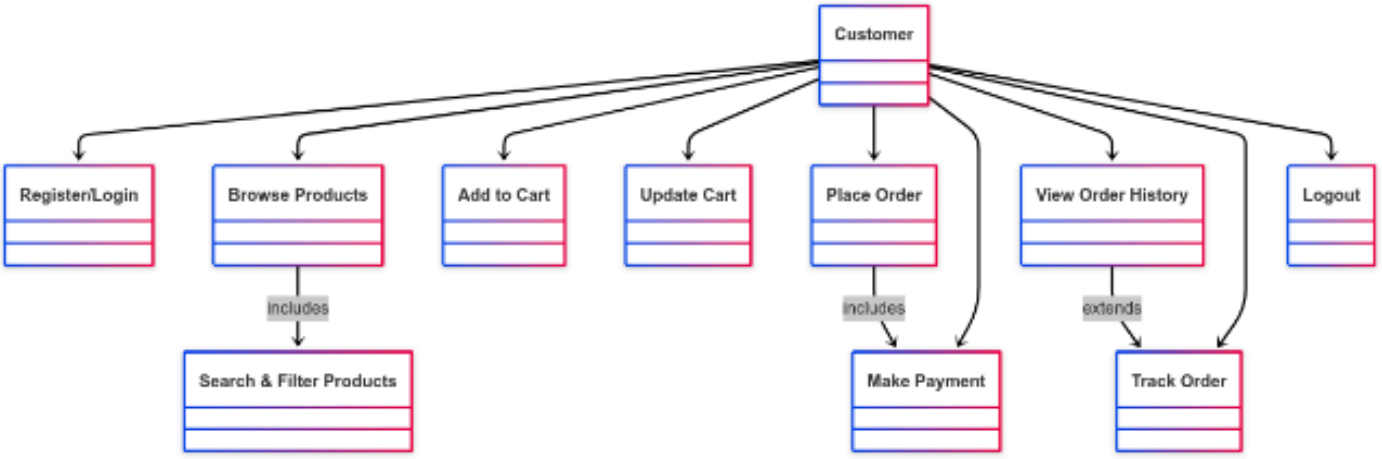
- **User Registration** – Allow users to create an account.
- **User Login** – Enable users to log in securely.
- **Browse Products** – Display available medicines and categories.
- **Search Products** – Let users search for specific items.
- **Add to Cart** – Add selected products to the shopping cart.
- **View Cart** – Show current cart contents and prices.
- **Update Cart** – Edit quantities or remove items from the cart.
- **Place Order** – Submit an order with shipping and payment details.
- **Order Confirmation** – Display success message after order placement.
- **View Orders** – Show order history and details.
- **Track Order** – Indicate current status of the order.
- **Admin Login** – Allow admin to access management features.
- **Manage Products** – Admin can add, edit, or delete products.
- **Manage Orders** – Admin can view and update order statuses.

NON-FUNCTIONAL REQUIREMENTS

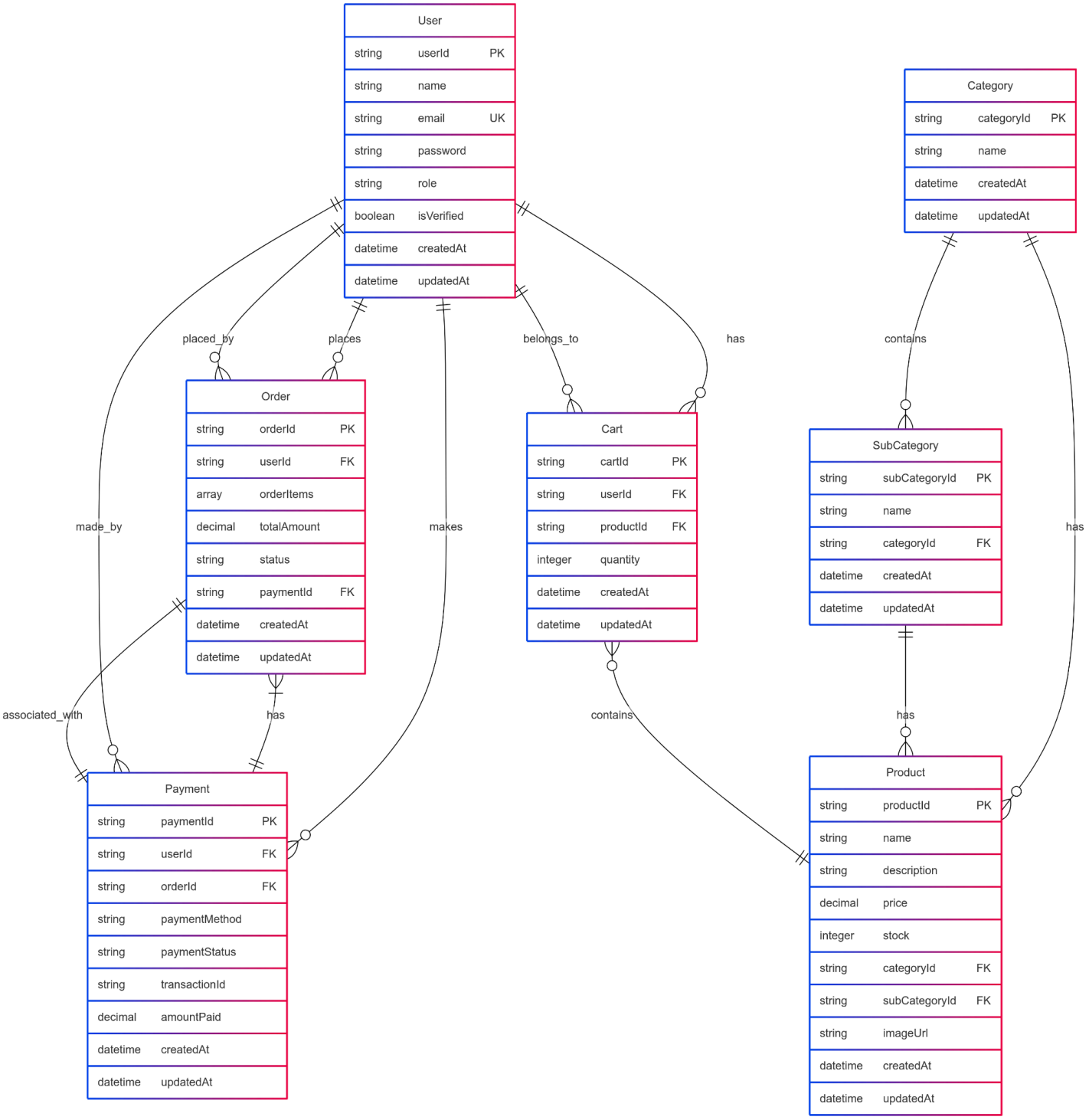
- **Performance** - Ensure fast page loading and transaction processing for a smooth user experience.
- **Security** - Protect user data, especially payment details, through encryption and secure storage.
- **Usab**ility - Provide an intuitive and user-friendly interface for both customers and admins.
- **Scalability** -Allow the system to handle increasing user traffic and order volume without performance degradation.
- **Reliability** - Ensure system stability, minimal downtime, and quick recovery from failures.

USE CASE DIAGRAM

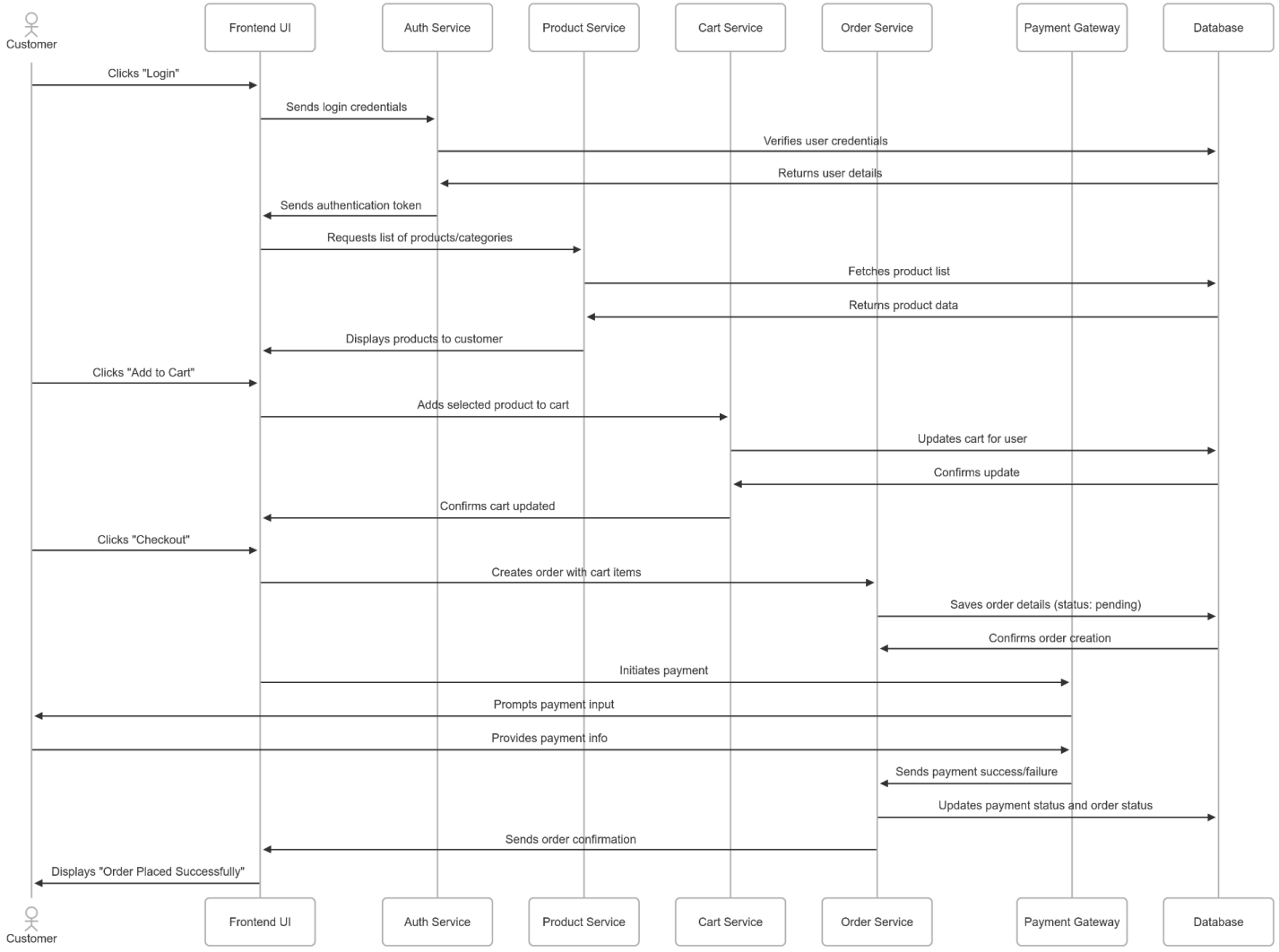



Customer
Register/Login
Browse Products
Add to Cart
Update Cart
Place Order
View Order History
Logout
includes
includes
extends
Search & Filter Products
Make Payment
Track Order

ER DIAGRAM


User
string userId PK
string name
string email UK
string password
string role
boolean isVerified
datetime createdAt
datetime updatedAt
Category
string categoryId PK
string name
datetime createdAt
datetime updatedAt
placed_by
places
belongs_to
has
contains
Order
string orderId PK
string userId FK
array orderItems
decimal totalAmount
string status
string paymentId FK
datetime createdAt
datetime updatedAt
Cart
string cartId PK
string userId FK
string productId FK
integer quantity
datetime createdAt
datetime updatedAt
SubCategory
string subCategoryId PK
string name
string categoryId FK
datetime createdAt
datetime updatedAt
made_by
makes
has
associated_with
has
contains
has
Payment
string paymentId PK
string userId FK
string orderId FK
string paymentMethod
string paymentStatus
string transactionId
decimal amountPaid
datetime createdAt
datetime updatedAt
Product
string productId PK
string name
string description
decimal price
integer stock
string categoryId FK
string subCategoryId FK
string imageUrl
datetime createdAt
datetime updatedAt

SEQUENCE DIAGRAM



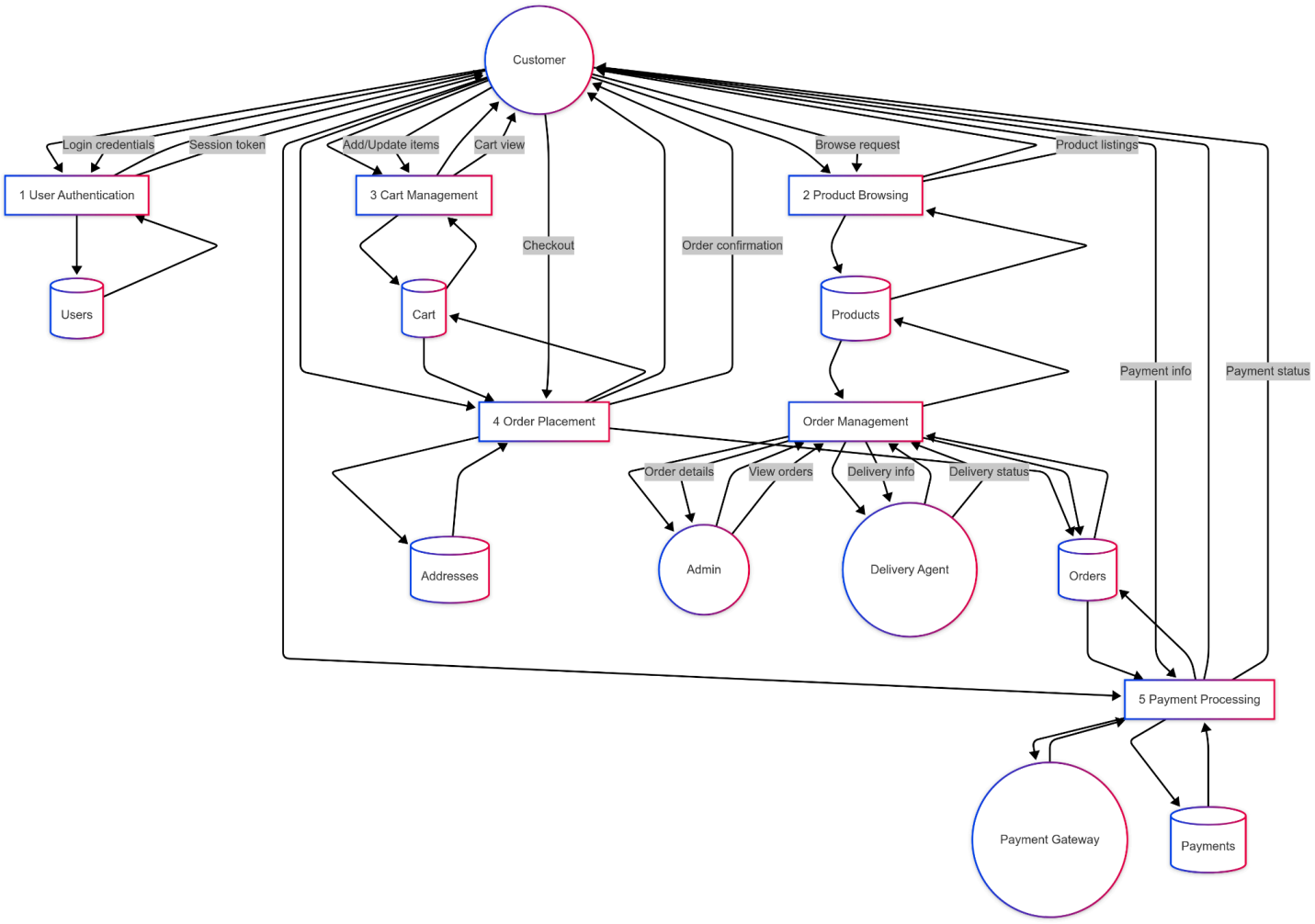
DFD (DATA FLOW DIAGRAM)

Diagram Level: Level 1 (Context-Level & Detailed)

Purpose: To represent the flow of data between external entities (like users or admin) and internal system processes (like order processing or product management) in the MedCart application.

The **Level 0 DFD** shows how data flows in and out of the MedCart system. The system interacts with two main external entities: **Customers** and **Admin**. Customers browse products, place orders, and make payments, while Admins manage products and handle order fulfillment. The system stores user profiles, order records, product inventory, and payment data.

The **Level 1 DFD** of MedCart provides a detailed view of the internal processes such as browsing products, placing orders, making payments, and admin management. It illustrates how each process interacts with corresponding data stores like Product Data, Order Data, and Payment Info to perform system functions.



FRONT-END CODE SNIPPET

COMPONENT NAME : CheckoutPage.jsx

This component typically handles:

- Showing items in the cart
- Capturing shipping address
- Selecting payment method (Stripe or COD)
- Calling the backend to create the order

Here's the code :

```
import React, { useState } from 'react'
import { useGlobalContext } from '../provider/GlobalProvider'
import { DisplayPriceInRupees } from '../utils/DisplayPriceInRupees'
import AddAddress from '../components/AddAddress'
import { useSelector } from 'react-redux'
import AxiosToastError from '../utils/AxiosToastError'
import Axios from '../utils/Axios'
import SummaryApi from '../common/SummaryApi'
import toast from 'react-hot-toast'
import { useNavigate } from 'react-router-dom'
import { loadStripe } from '@stripe/stripe-js'

const CheckoutPage = () => {
  const { notDiscountTotalPrice, totalPrice, totalQty, fetchCartItem,fetchOrder } = useGlobalContext()
```

```
const [openAddress, setOpenAddress] = useState(false)
const addressList = useSelector(state => state.addresses.addressList)
const [selectAddress, setSelectAddress] = useState(0)
const cartItemsList = useSelector(state => state.cartItem.cart)
const navigate = useNavigate()

const handleCashOnDelivery = async() => {
   try {
      const response = await Axios({
        ...SummaryApi.CashOnDeliveryOrder,
        data : {
          list_items : cartItemsList,
          addressId : addressList[selectAddress]?._id,
          subTotalAmt : totalPrice,
          totalAmt :  totalPrice,
        }
      })

      const { data : responseData } = response

      if(responseData.success){
          toast.success(responseData.message)
          if(fetchCartItem){
            fetchCartItem()
          }
          if(fetchOrder){
            fetchOrder()
          }
          navigate('/success',{
            state : {
              text : "Order"
            }
          })
      }

   } catch (error) {
     AxiosToastError(error)
   }
}

const handleOnlinePayment = async()=>{
  try {
     toast.loading("Loading...")
     const stripePublicKey = import.meta.env.VITE_STRIPE_PUBLIC_KEY
     const stripePromise = await loadStripe(stripePublicKey)

     const response = await Axios({
        ...SummaryApi.payment_url,
        data : {
          list_items : cartItemsList,
```

```
            addressId : addressList[selectAddress]?._id,
            subTotalAmt : totalPrice,
            totalAmt :  totalPrice,
          }
        })

        const { data : responseData } = response

        stripePromise.redirectToCheckout({ sessionId : responseData.id })

        if(fetchCartItem){
          fetchCartItem()
        }
        if(fetchOrder){
          fetchOrder()
        }
      } catch (error) {
        AxiosToastError(error)
      }
    }
    return (
      <section className='bg-blue-50'>
        <div className='container mx-auto p-4 flex flex-col lg:flex-row w-full gap-5 justify-between'>
          <div className='w-full'>
            {/***address***/}
            <h3 className='text-lg font-semibold'>Choose your address</h3>
            <div className='bg-white p-2 grid gap-4'>
              {
                addressList.map((address, index) => {
                  return (
                    <label htmlFor={"address" + index} className={!address.status && "hidden"}>
                      <div className='border rounded p-3 flex gap-3 hover:bg-blue-50'>
                        <div>
                          <input id={"address" + index} type='radio' value={index} onChange={(e) =>
setSelectAddress(e.target.value)} name='address' />
                        </div>
                        <div>
                          <p>{address.address_line}</p>
                          <p>{address.city}</p>
                          <p>{address.state}</p>
                          <p>{address.country} - {address.pincode}</p>
                          <p>{address.mobile}</p>
                        </div>
                      </div>
                    </label>
                  )
                })
              }
              <div onClick={() => setOpenAddress(true)} className='h-16 bg-blue-50 border-2 border-dashed flex
justify-center items-center cursor-pointer'>
```

```
            Add address
          </div>
        </div>



      </div>

      <div className='w-full max-w-md bg-white py-4 px-2'>
        {/**summary**/}
        <h3 className='text-lg font-semibold'>Summary</h3>
        <div className='bg-white p-4'>
          <h3 className='font-semibold'>Bill details</h3>
          <div className='flex gap-4 justify-between ml-1'>
            <p>Items total</p>
            <p className='flex items-center gap-2'><span className='line-through
text-neutral-400'>{DisplayPriceInRupees(notDiscountTotalPrice)}</span><span>{DisplayPriceInRupees(total
Price)}</span></p>
          </div>
          <div className='flex gap-4 justify-between ml-1'>
            <p>Quntity total</p>
            <p className='flex items-center gap-2'>{totalQty} item</p>
          </div>
          <div className='flex gap-4 justify-between ml-1'>
            <p>Delivery Charge</p>
            <p className='flex items-center gap-2'>Free</p>
          </div>
          <div className='font-semibold flex items-center justify-between gap-4'>
            <p >Grand total</p>
            <p>{DisplayPriceInRupees(totalPrice)}</p>
          </div>
        </div>
        <div className='w-full flex flex-col gap-4'>
          <button className='py-2 px-4 bg-green-600 hover:bg-green-700 rounded text-white font-semibold'
onClick={handleOnlinePayment}>Online Payment</button>

          <button className='py-2 px-4 border-2 border-green-600 font-semibold text-green-600
hover:bg-green-600 hover:text-white' onClick={handleCashOnDelivery}>Cash on Delivery</button>
        </div>
      </div>
    </div>



    {
      openAddress && (
        <AddAddress close={() => setOpenAddress(false)} />
      )
    }
  </section>
)
```

```
}

export default CheckoutPage
```

BACKEND CODE SNIPPET

CONTROLLER NAME : order.controller.js

This controller typically handles:

- The order is created (for both COD and Stripe)
- It interacts with MongoDB (via Mongoose models)
- It validates input and handles errors
- It returns a success/failure response

Here's the code :

```
import Stripe from "../config/stripe.js";
import CartProductModel from "../models/cartproduct.model.js";
import OrderModel from "../models/order.model.js";
import UserModel from "../models/user.model.js";
import mongoose from "mongoose";

 export async function CashOnDeliveryOrderController(request,response){
   try {
      const userId = request.userId // auth middleware
      const { list_items, totalAmt, addressId,subTotalAmt } = request.body

      const payload = list_items.map(el => {
         return({
            userId : userId,
            orderId : `ORD-${new mongoose.Types.ObjectId()}`,
            productId : el.productId._id,
            product_details : {
               name : el.productId.name,
               image : el.productId.image
            } ,
            paymentId : "",
            payment_status : "CASH ON DELIVERY",
            delivery_address : addressId ,
            subTotalAmt  : subTotalAmt,
            totalAmt  :  totalAmt,
         })
      })

      const generatedOrder = await OrderModel.insertMany(payload)

      ///remove from the cart
      const removeCartItems = await CartProductModel.deleteMany({ userId : userId })
      const updateInUser = await UserModel.updateOne({ _id : userId }, { shopping_cart : []})
```

```
        return response.json({
            message : "Order successfully",
            error : false,
            success : true,
            data : generatedOrder
        })

    } catch (error) {
        return response.status(500).json({
            message : error.message || error ,
            error : true,
            success : false
        })
    }
}

export const pricewithDiscount = (price,dis = 1)=>{
    const discountAmout = Math.ceil((Number(price) * Number(dis)) / 100)
    const actualPrice = Number(price) - Number(discountAmout)
    return actualPrice
}

export async function paymentController(request,response){
    try {
        const userId = request.userId // auth middleware
        const { list_items, totalAmt, addressId,subTotalAmt } = request.body

        const user = await UserModel.findById(userId)

        const line_items  = list_items.map(item =>{
            return{
              price_data : {
                  currency : 'inr',
                  product_data : {
                      name : item.productId.name,
                      images : item.productId.image,
                      metadata : {
                          productId : item.productId._id
                      }
                  },
                  unit_amount : pricewithDiscount(item.productId.price,item.productId.discount) * 100
              },
              adjustable_quantity : {
                  enabled : true,
                  minimum : 1
              },
              quantity : item.quantity
            }
        })
```

```
        const params = {
            submit_type : 'pay',
            mode : 'payment',
            payment_method_types : ['card'],
            customer_email : user.email,
            metadata : {
                userId : userId,
                addressId : addressId
            },
            line_items : line_items,
            success_url : `${process.env.FRONTEND_URL}/success`,
            cancel_url : `${process.env.FRONTEND_URL}/cancel`
        }

        const session = await Stripe.checkout.sessions.create(params)

        return response.status(200).json(session)

    } catch (error) {
        return response.status(500).json({
            message : error.message || error,
            error : true,
            success : false
        })
    }
}


const getOrderProductItems = async({
    lineItems,
    userId,
    addressId,
    paymentId,
    payment_status,
 })=>{
    const productList = []

    if(lineItems?.data?.length){
        for(const item of lineItems.data){
            const product = await Stripe.products.retrieve(item.price.product)

            const paylod = {
                userId : userId,
                orderId : `ORD-${new mongoose.Types.ObjectId()}`,
                productId : product.metadata.productId,
                product_details : {
                    name : product.name,
                    image : product.images
                } ,
```

```
            paymentId : paymentId,
            payment_status : payment_status,
            delivery_address : addressId,
            subTotalAmt  : Number(item.amount_total / 100),
            totalAmt  :  Number(item.amount_total / 100),
          }

          productList.push(paylod)
        }
    }

    return productList
}

//http://localhost:8080/api/order/webhook
export async function webhookStripe(request,response){
    const event = request.body;
    const endPointSecret = process.env.STRIPE_ENPOINT_WEBHOOK_SECRET_KEY

    console.log("event",event)

    // Handle the event
  switch (event.type) {
    case 'checkout.session.completed':
      const session = event.data.object;
      const lineItems = await Stripe.checkout.sessions.listLineItems(session.id)
      const userId = session.metadata.userId
      const orderProduct = await getOrderProductItems(
        {
          lineItems : lineItems,
          userId : userId,
          addressId : session.metadata.addressId,
          paymentId  : session.payment_intent,
          payment_status : session.payment_status,
        })

      const order = await OrderModel.insertMany(orderProduct)

        console.log(order)
        if(Boolean(order[0])){
          const removeCartItems = await  UserModel.findByIdAndUpdate(userId,{
            shopping_cart : []
          })
          const removeCartProductDB = await CartProductModel.deleteMany({ userId : userId})
        }
      break;
    default:
      console.log(`Unhandled event type ${event.type}`);
  }
```

```
  // Return a response to acknowledge receipt of the event
  response.json({received: true});
}


export async function getOrderDetailsController(request,response){
   try {
      const userId = request.userId // order id

      const orderlist = await OrderModel.find({ userId : userId }).sort({ createdAt : -1
}).populate('delivery_address')

      return response.json({
         message : "order list",
         data : orderlist,
         error : false,
         success : true
      })
   } catch (error) {
      return response.status(500).json({
         message : error.message || error,
         error : true,
         success : false
      })
   }
}
```

USE CASE TEST

Use Case ID: UC-004

Use Case Name: Checkout and Place Order

Actor(s): Registered User

Description:

This use case describes how a user checks out items from their cart using either Cash on Delivery or Online Payment, resulting in the placement of an order and clearing of the cart.

Preconditions:

- The user is logged into the system.
- The cart contains one or more items.
- The user has added at least one delivery address.

Main Flow:

- User navigates to the Checkout Page.
- The system displays cart summary and saved addresses.
- The user selects an address.

- User selects a payment method:
    - If Cash on Delivery:
        - Order details are sent to the backend via CashOnDeliveryOrderController.
        - Cart items are saved to the Order DB.
        - Cart is cleared.
        - User is redirected to the success page.
    - If Online Payment:
        - Stripe Checkout session is created
        - After successful payment, webhook triggers (webhookStripe).
        - Order is saved and cart is cleared.
        - User is redirected to the success page.

Postconditions

- An order entry is created in the database.
- User cart is emptied.
- The order appears in the user's order history.

Test Data Example:

- Cart Items: 2 (Detol, Baby Diaper)
- Address: "12, Park Street, Kolkata"
- Payment Method: Online Payment

Expected Result:

- Stripe session redirects user to payment.
- On success, order is created.
- Cart is cleared.
- Success message shown.

GITHUB REPOSITORY LINK : https://github.com/arundhuti06/MedCart