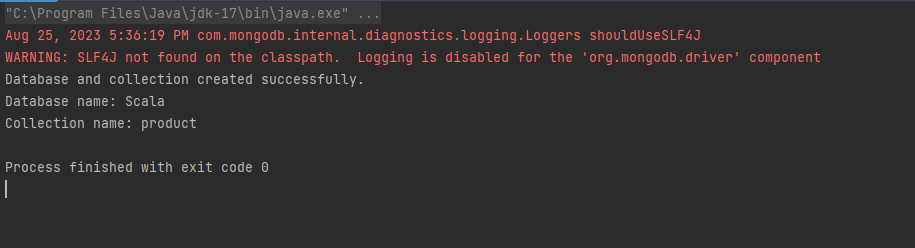
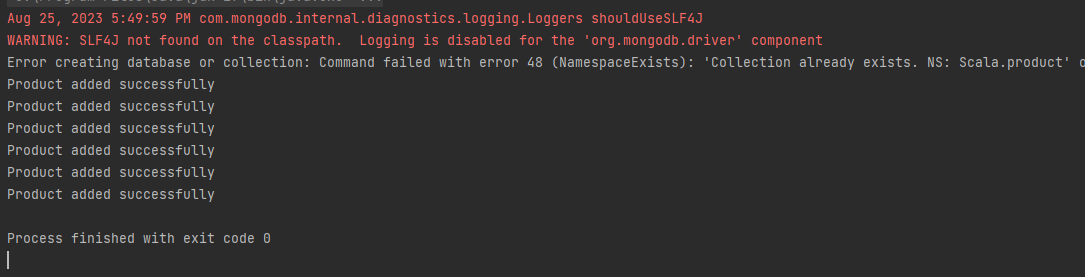
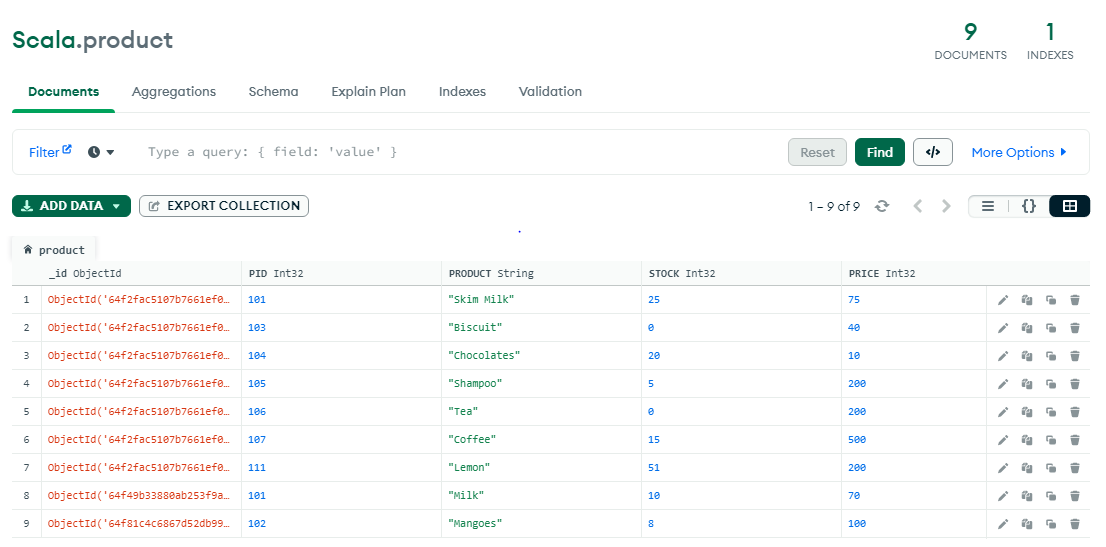
**Supermarket Management System using MongoDB**

1. **Database Connectivity**

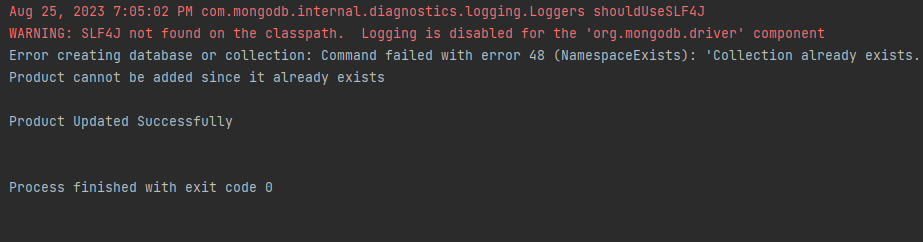
****

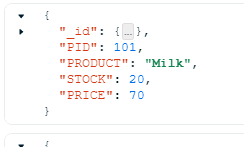
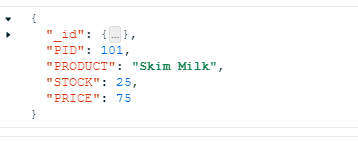
1. **Inserting Product**

****

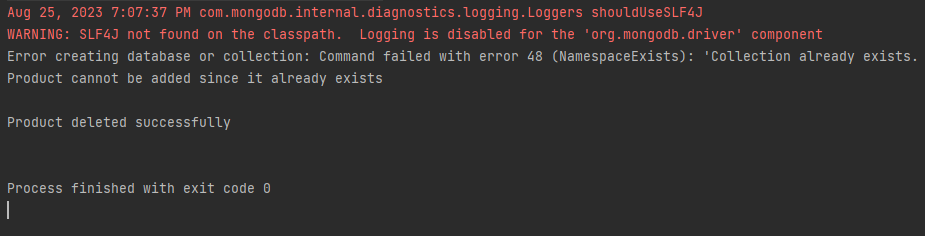
****

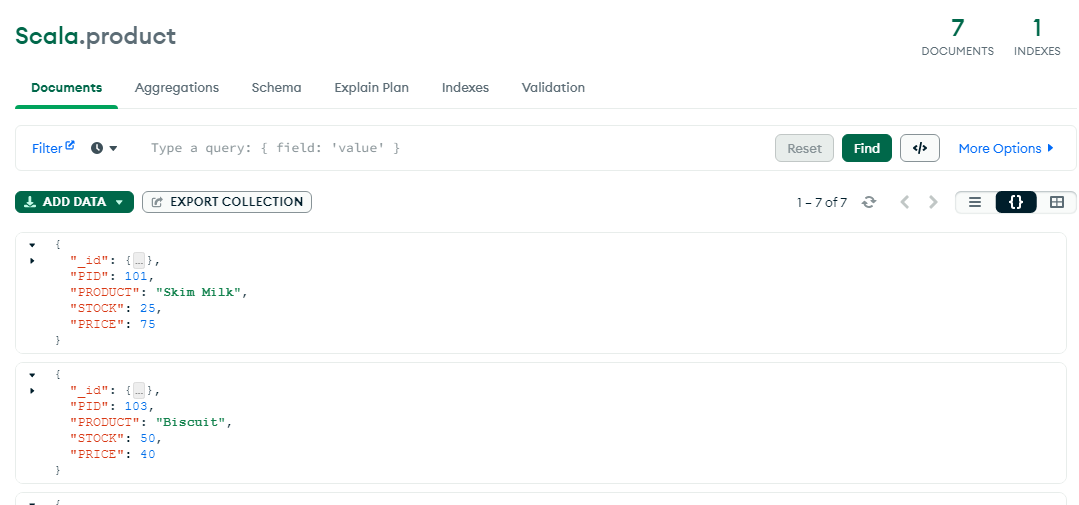
1. **Updating a product**

****

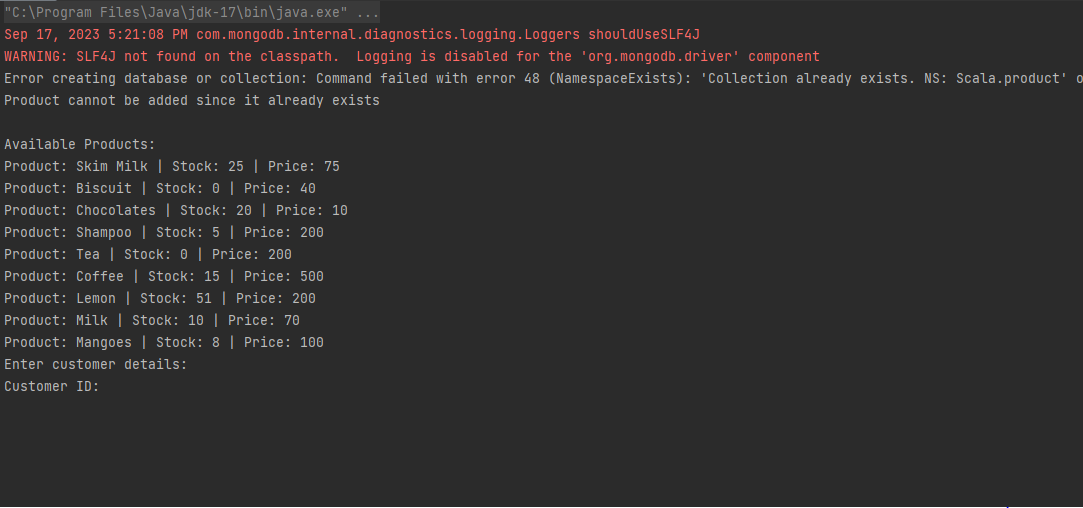
**🡪 **

1. **Deleting a product (101 - milk)**

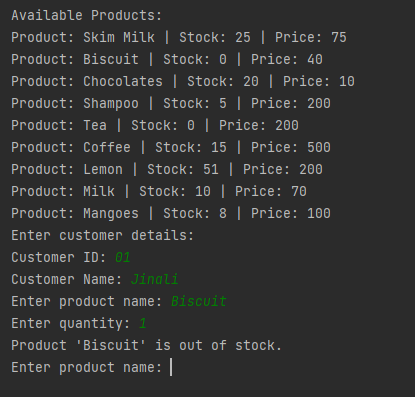
****

****

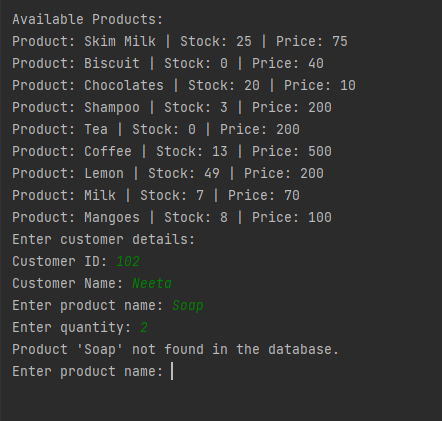
1. **Customer management**
2. **Displaying products for customers**
3. **User input for purchasing**

****

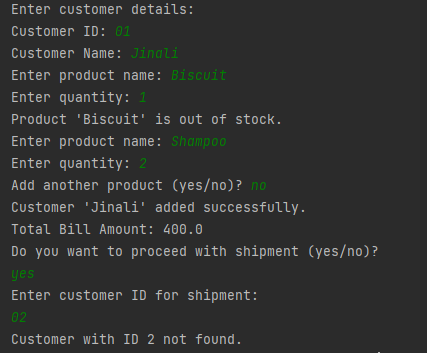
1. **Out of stock if the stock value zero**

****

1. **Display product not found if not available:**

****

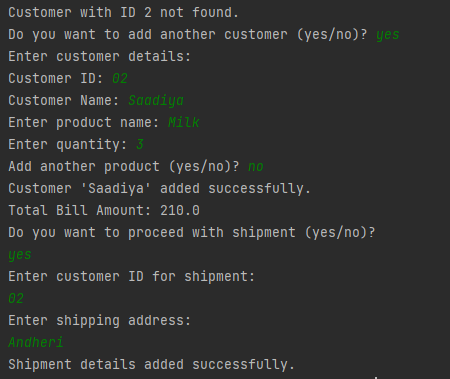
1. **Total bill is generated if no further product is required**
2. **Shipment Management if required**

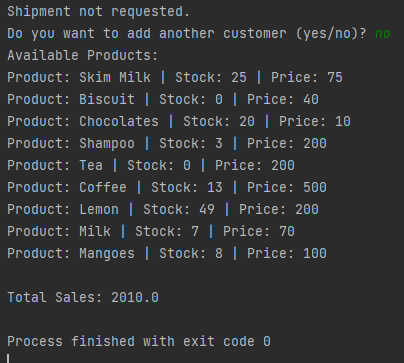
****

**Stock getting reduced from the database**

****

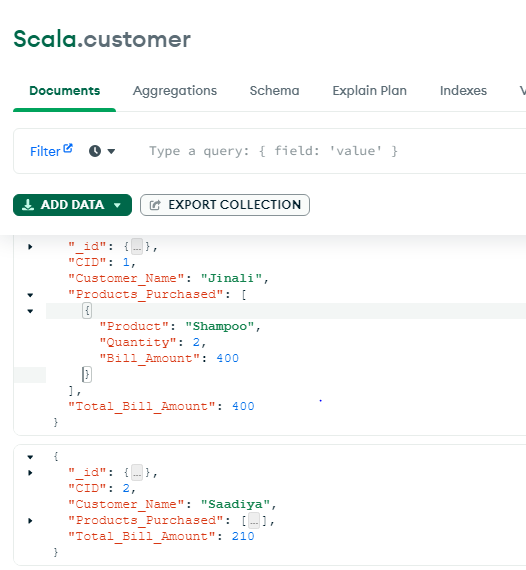
****

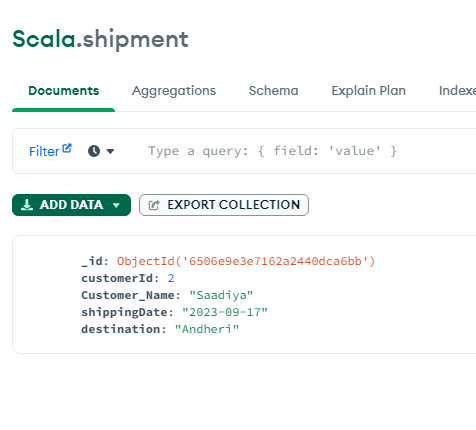
****

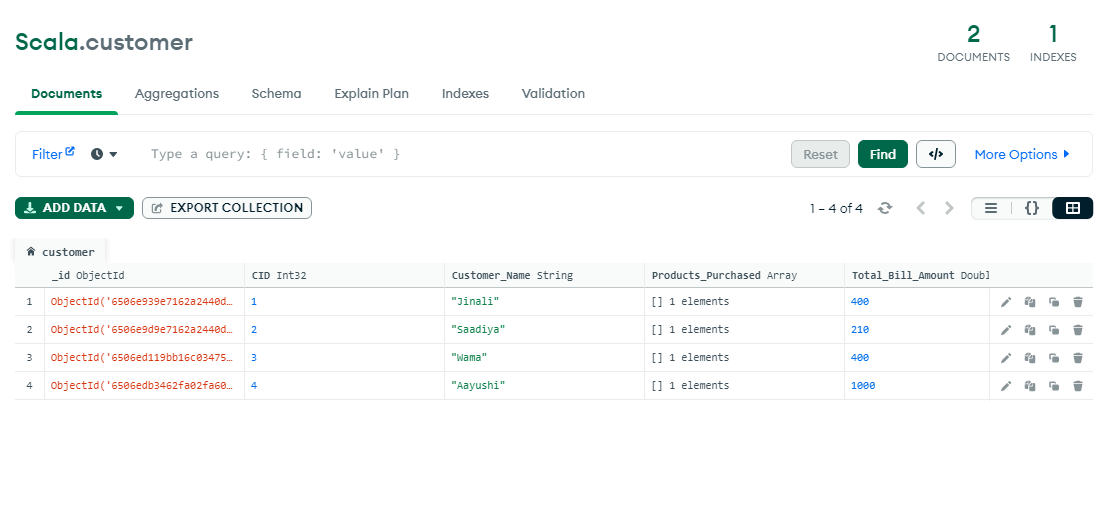
****

**Finally displaying the products available after the products purchased by the customer**

**And displaying the total sales at the end of the day.**







**Code:**

import org.mongodb.scala.MongoClient  
  
import scala.concurrent.Await  
import scala.concurrent.duration.DurationInt  
import org.mongodb.scala.Document  
import scala.math.Numeric.DoubleIsFractional  
import org.mongodb.scala.bson.conversions.Bson  
import org.mongodb.scala.bson.{BsonArray, ObjectId}  
import org.mongodb.scala.model.Filters  
  
import scala.jdk.CollectionConverters.CollectionHasAsScala  
  
  
object Main extends App {  
 val *mongoUrl* = "mongodb://localhost:27017"  
 val *mongoClient* = *MongoClient*(*mongoUrl*)  
 val *database* = *mongoClient*.getDatabase("Scala")  
 val *collection* = *database*.getCollection("product")  
 val *customerCollection* = *database*.getCollection("customer") //for customer management  
 val *shipmentCollection* = *database*.getCollection("shipment") // shipment  
  
 try {  
 Await.*result*(*database*.createCollection("product").toFuture(), 100.seconds)  
  
 *println*("Database and collection created successfully.")  
 *println*(s"Database name: **$**{*database*.*name*}")  
 *println*(s"Collection name: **$**{*collection*.*namespace*.getCollectionName}")  
 } catch {  
 case ex: Throwable =>  
 *println*(s"Error creating database or collection: **$**{ex.getMessage}")  
 }  
  
  
 // insert code  
  
 val *documentsToInsert* = Seq(  
 *Document*("PID" -> 101, "PRODUCT" -> "Milk", "STOCK" -> 20, "PRICE" -> 70)  
 // Document("PID" -> 102, "PRODUCT" -> "Mangoes", "STOCK" -> 24, "PRICE" -> 100),  
 // Document("PID" -> 103, "PRODUCT" -> "Biscuit", "STOCK" -> 50, "PRICE" -> 40),  
 // Document("PID" -> 104, "PRODUCT" -> "Chocolates", "STOCK" -> 400, "PRICE" -> 10),  
 // Document("PID" -> 105, "PRODUCT" -> "Shampoo", "STOCK" -> 10, "PRICE" -> 200),  
 // Document("PID" -> 106, "PRODUCT" -> "Tea", "STOCK" -> 10, "PRICE" -> 200),  
 // Document("PID" -> 107, "PRODUCT" -> "Coffee", "STOCK" -> 30, "PRICE" -> 500),  
 // Document("PID" -> 111, "PRODUCT" -> "Lemon", "STOCK" -> 10, "PRICE" -> 200)  
  
 )  
  
 *documentsToInsert*.foreach { document =>  
 val existingProduct = Await.*result*(*collection*.find(*Document*("PID" -> document("PID"), "PRODUCT" -> document("PRODUCT"))).toFuture(), 100.seconds)  
  
 if (existingProduct.isEmpty) {  
 Await.*result*(*collection*.insertOne(document).toFuture(), 100.seconds)  
 *println*("Product added successfully")  
 } else {  
 *println*("Product cannot be added since it already exists")  
 //println()  
 }  
 }  
  
 val *result* = Await.*result*(*collection*.find().toFuture(), 100.seconds)  
 //println(tabulate.doc.docInstances.print(result))  
  
 //read operation  
 def printAllDocuments(): Unit = {  
 val result = Await.*result*(*collection*.find().toFuture(), 100.seconds)  
 *println*()  
 result.foreach(*println*)  
 }  
 //printAllDocuments()  
  
 // Update Data  
 def updateProduct(pid: Int, updatedDocument: Document): Unit = {  
 val query = *Document*("PID" -> pid)  
 Await.*result*(*collection*.replaceOne(query, updatedDocument).toFuture(), 100.seconds)  
 *println*()  
 *println*("Product Updated Successfully")  
 *println*()  
 }  
  
 //val updateDocument = Document("PID" -> 101, "PRODUCT" -> "Skim Milk", "STOCK" -> 25, "PRICE" -> 75)  
 //updateProduct(101, updateDocument)  
  
 // Delete data  
  
 def deleteProduct(pid: Int): Unit = {  
 val query = *Document*("PID" -> pid)  
 Await.*result*(*collection*.deleteOne(query).toFuture(), 100.seconds)  
 *println*("Product deleted successfully")  
 *println*()  
 }  
 //deleteProduct(102)  
  
 // sample example:  
  
 // 2) read  
 // printAllDocuments()  
 *println*()  
  
 // 3) update  
 // val updateDocument = Document("PID" -> 101, "PRODUCT" -> "Skim Milk", "STOCK" -> 25, "PRICE" -> 75)  
 // updateProduct(101, updateDocument)  
  
 // 4) delete  
 // deleteProduct(102)  
  
 // printing all after crud operations  
 //printAllDocuments()  
  
 // Function to list all available products  
 def listAvailableProducts(): Unit = {  
  
 val availableProducts = Await.*result*(*collection*.find().toFuture(), 100.seconds)  
  
 // Check if availableProducts is not empty.  
 if (availableProducts.nonEmpty) {  
  
 *println*("Available Products:")  
  
 // Iterate through each product in the availableProducts list.  
 availableProducts.foreach { product =>  
  
 // Extract the product name from the current product document.  
 val productName = product.getString("PRODUCT")  
  
 // Extract the stock quantity from the current product document.  
 val stock = product.getInteger("STOCK")  
  
 // Extract the price from the current product document.  
 val price = product.getInteger("PRICE")  
  
 // Print the product information.  
 *println*(s"Product: **$**productName | Stock: **$**stock | Price: **$**price")  
 }  
 } else {  
  
 *println*("No products available in the market.")  
 }  
 }  
 *listAvailableProducts*()  
  
  
 // Customer management :  
  
  
 // Define a function to manage customer interactions and generate a bill.  
 def manageCustomerAndGenerateBill(): Unit = {  
  
 *println*("Enter customer details:")  
 *print*("Customer ID: ")  
 val customerId = scala.io.StdIn.readInt()  
 *print*("Customer Name: ")  
 val customerName = scala.io.StdIn.readLine()  
  
 // Initialize an empty list to store product details for the bill.  
 var productsPurchased = List.*empty*[Document]  
 // Initialize a flag to control the loop.  
 var continue = true  
  
 // Start a loop for adding products to the bill.  
 while (continue) {  
  
 *print*("Enter product name: ")  
 val productName = scala.io.StdIn.readLine()  
  
 *print*("Enter quantity: ")  
 val quantity = scala.io.StdIn.readInt()  
  
 // Create a query document to search for the product in the database.  
 val productQuery = *Document*("PRODUCT" -> productName)  
 // Query the database for the product and wait for the result.  
 val existingProduct = Await.*result*(*collection*.find(productQuery).toFuture(), 100.seconds).headOption  
  
 // Check if the product exists in the database.  
 if (existingProduct.isDefined) {  
 val product = existingProduct.get  
 // Get the current stock quantity of the product.  
 val currentStock = product.getInteger("STOCK")  
  
 // Check if there is enough stock for the requested quantity.  
 if (currentStock >= quantity) {  
 val updatedStock = currentStock - quantity  
  
 // Create an update query to reduce the stock quantity.  
 val updateQuery = *Document*("PRODUCT" -> productName)  
 // Update the stock quantity in the database.  
 val updateDocument = *Document*("$set" -> *Document*("STOCK" -> updatedStock))  
 Await.*result*(*collection*.updateOne(updateQuery, updateDocument).toFuture(), 100.seconds)  
  
 // Get the price of the product.  
 val price = product.getInteger("PRICE")  
 // Calculate the bill amount for the current product.  
 val billAmount = price.toDouble \* quantity  
 // Create a document to represent the purchased product.  
 val productDocument = *Document*(  
 "Product" -> productName,  
 "Quantity" -> quantity,  
 "Bill\_Amount" -> billAmount  
 )  
 // Add the product to the list of purchased products.  
 productsPurchased = productsPurchased :+ productDocument  
  
 *print*("Add another product (yes/no)? ")  
 val input = scala.io.StdIn.readLine().toLowerCase  
  
 if (input != "yes") {  
 continue = false  
 }  
 } else {  
  
 *println*(s"Product '**$**productName' is out of stock.")  
 }  
 } else {  
 *println*(s"Product '**$**productName' not found in the database.")  
 }  
 }  
  
 // Calculate the total bill amount for all purchased products.  
 val totalBillAmount = productsPurchased.map(\_.getDouble("Bill\_Amount")).foldLeft(0.0)(\_ + \_)  
  
  
  
 // Create a document to represent the customer and their bill details.  
 val customerDocument = *Document*(  
 "CID" -> customerId,  
 "Customer\_Name" -> customerName,  
 "Products\_Purchased" -> productsPurchased,  
 "Total\_Bill\_Amount" -> totalBillAmount  
 )  
  
 Await.*result*(*customerCollection*.insertOne(customerDocument).toFuture(), 100.seconds)  
  
 *println*(s"Customer '**$**customerName' added successfully.")  
 *println*(s"Total Bill Amount: **$**totalBillAmount")  
 }  
  
 *manageCustomerAndGenerateBill*()  
  
  
  
  
  
  
  
 // Shipment code  
  
 // Ask the user if they want to proceed with shipment  
 def manageShipment(): Unit = {  
 *println*("Do you want to proceed with shipment (yes/no)?")  
 val shipmentChoice = scala.io.StdIn.readLine().toLowerCase  
  
 if (shipmentChoice == "yes") {  
 *println*("Enter customer ID for shipment:")  
 val customerIdForShipment = scala.io.StdIn.readInt()  
  
 val customerQuery = *Document*("CID" -> customerIdForShipment)  
 val existingCustomer = Await.*result*(*customerCollection*.find(customerQuery).toFuture(), 100.seconds).headOption  
  
 if (existingCustomer.isDefined) {  
 // shipment details  
 *println*("Enter shipping address:")  
 val shippingAddress = scala.io.StdIn.readLine()  
 val shippingDate = java.time.LocalDate.*now*.toString // Get the current date  
  
 val shipmentDocument = *Document*(  
 // "shipmentId" -> shipmentId,  
 "customerId" -> customerIdForShipment,  
 "Customer\_Name" -> existingCustomer.get.getString("Customer\_Name"),  
 "shippingDate" -> shippingDate,  
 "destination" -> shippingAddress  
 )  
  
 // Add the shipment to the shipment collection  
 Await.*result*(*shipmentCollection*.insertOne(shipmentDocument).toFuture(), 100.seconds)  
  
 *println*("Shipment details added successfully.")  
 } else {  
 *println*(s"Customer with ID **$**customerIdForShipment not found.")  
 }  
 } else {  
 *println*("Shipment not requested.")  
 }  
  
 }  
  
 // Call the shipment management function  
 // Ask the user if they want to proceed with shipment  
 *manageShipment*()  
  
  
 /// Function to interactively manage customers and generate bills  
 def manageCustomersInteractively(): Unit = {  
 var continue = true  
  
 while (continue) {  
 *print*("Do you want to add another customer (yes/no)? ")  
 val input = scala.io.StdIn.readLine()  
  
  
 if (input == "yes") {  
 *manageCustomerAndGenerateBill*()  
 *manageShipment*()  
  
 } else if (input == "no") {  
 continue = false  
 }  
 }  
 }  
   
 // Start customer management and bill generation  
 *manageCustomersInteractively*()  
  
 // List available products  
 *listAvailableProducts*()

// Calculate total sales  
val *totalSales* = Await.*result*(*customerCollection*.find().toFuture(), 100.seconds)  
 .map(\_.getDouble("Total\_Bill\_Amount"))  
 .foldLeft(0.0)(\_ + \_)  
*println*()  
*println*(s"Total Sales: **$***totalSales*")

}