USE CASE STUDY REPORT

RETAIL STORE INVENTORY MANAGEMENT SYSTEM

Group No.: Group 7

STUDENT NAME: ROHAN PRAKASH KRISHNA PRAKASH & NISHCHAY LINGE GOWDA

Executive Summary:

A state-of-the-art solution created to handle the intricate problems of contemporary retail operations is the Retail Store Inventory Management System. This all-inclusive solution seeks to boost supply chain effectiveness, streamline inventory procedures, and boost overall company performance. The system reduces overstocking and stockouts by providing accurate, current information on stock levels using real-time data tracking, advanced analytics, and automation. This reduces the expense of keeping inventory on hand and guarantees steady product availability, which raises consumer satisfaction. When stock hits certain criteria, an automated reordering procedure at the system's core initiates purchase orders.

This feature guarantees that necessary products are always available while reducing costs when paired with sophisticated analytics that examine purchase patterns, sales velocity, and supply chain logistics. human error in the process of placing new orders. Managers can react swiftly to shifting consumer tastes and market demands because to the system's capacity to produce actionable insights for inventory optimization, which promotes data-driven decision-making throughout the company.

Additionally, the Retail Store Inventory Management System facilitates smooth integration with current supply chain processes, which expedites supplier and retailer communications. Easy inventory management and monitoring across several sites is made possible by this integration in conjunction with an intuitive user interface with customizable dashboards and reports. Retail companies that use this technology should anticipate major increases in operational efficiency, a decrease in human labor, higher profitability due to optimal stock levels, and improved forecasting skills for future inventory requirements. Given how the retail industry is changing, this technology is a wise investment that puts businesses in a position for long-term expansion and a competitive edge.

I. Introduction

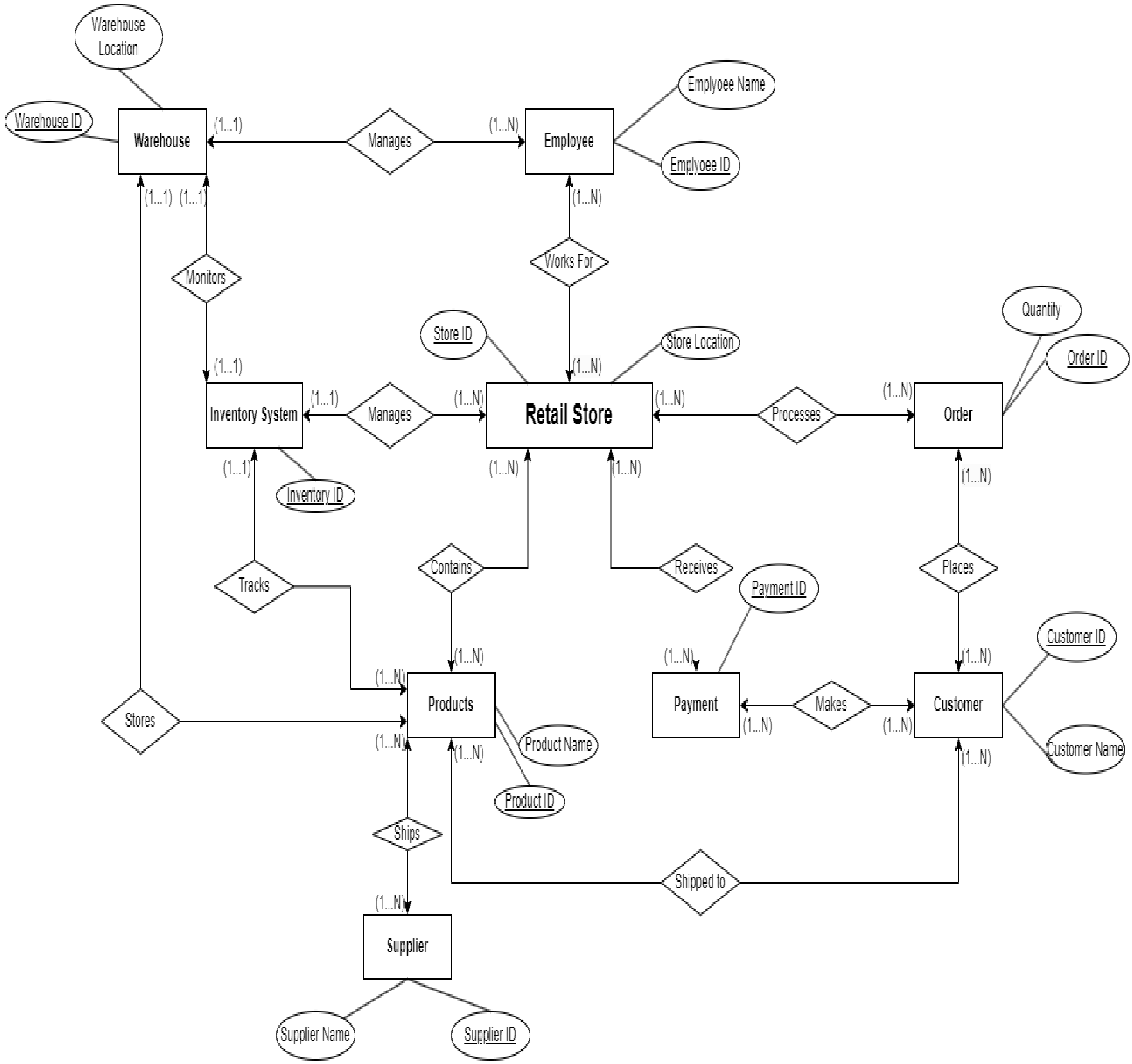
The Retail Store Inventory Management System is a complete system made to handle the intricate problems that contemporary retail businesses encounter. With the growth of retail companies and the changing expectations of consumers, effective inventory management has become critical. Automation, real-time data collecting, and advanced analytics are just a few of the cutting-edge technology that this system integrates to streamline inventory procedures. Accurately monitoring inventory levels, orders, sales, and delivery helps the system minimize overstocking and stockouts by guaranteeing that the proper products are available to satisfy client requests. The system primarily uses supply chain logistics, sales velocity, and purchasing trends to keep a balanced stock flow and reduce human error in inventory management.

The solution improves supply chain efficiency and guarantees that necessary products are always available by using automated reordering procedures that are initiated by predetermined stock criteria. Store managers are empowered to make well-informed decisions based on precise, current data when they receive real-time updates on stock movements. In addition to raising client satisfaction, this degree of accuracy and responsiveness makes business operations run more smoothly and boosts profitability. This solution puts retail establishments in a strong position to prosper in a market that is becoming more competitive and dynamic by addressing the essential components of inventory management.

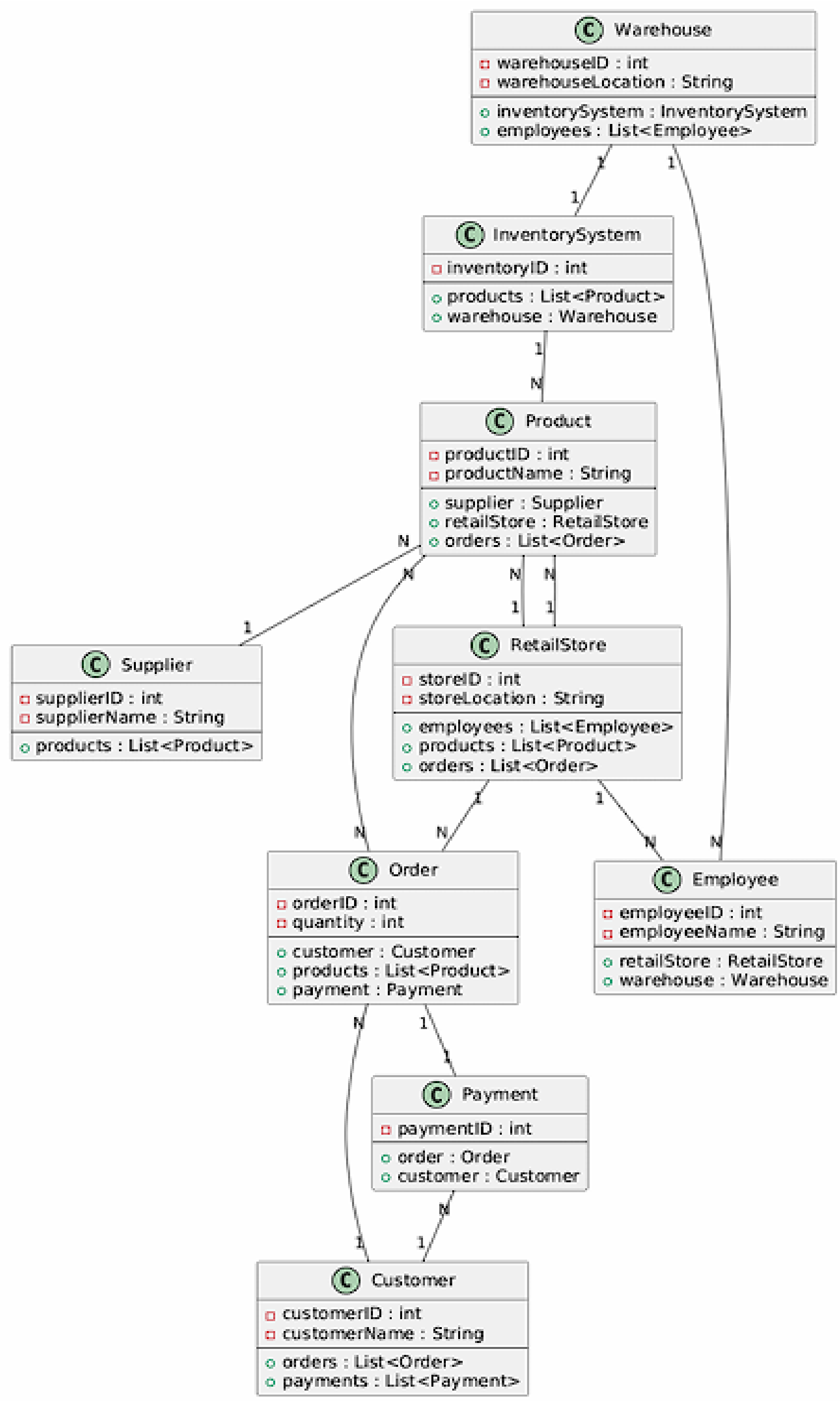
Retail companies should anticipate major increases in operational efficiency, a decrease in human labor, higher profitability through optimal stock levels, and improved forecasting skills for future inventory requirements by putting this system into place. This system is a strategic investment that puts businesses in a position for long-term growth and competitive advantage as the retail industry changes.

II. Conceptual Data Modeling

1. EER MODEL



2. UML MODEL


Warehouse
warehouseID : int
warehouseLocation : String
inventorySystem : InventorySystem
employees : List<Employee>
InventorySystem
inventoryID : int
products : List<Product>
warehouse : Warehouse
Product
productID : int
productName : String
supplier : Supplier
retailStore : RetailStore
orders : List<Order>
Supplier
supplierID : int
supplierName : String
products : List<Product>
RetailStore
storeID : int
storeLocation : String
employees : List<Employee>
products : List<Product>
orders : List<Order>
Order
orderID : int
quantity : int
customer : Customer
products : List<Product>
payment : Payment
Employee
employeeID : int
employeeName : String
retailStore : RetailStore
warehouse : Warehouse
Payment
paymentID : int
order : Order
customer : Customer
Customer
customerID : int
customerName : String
orders : List<Order>
payments : List<Payment>

III. Mapping Conceptual Model to Relational Model

Primary Key- Underlined **Foreign Key- *Italicized***

Customer (<u>Customer ID,</u> Customer Name)
Order (<u>Order ID</u>, Quantity)
Processes(*Order ID, Store ID*)

Order ID: foreign key refers to order ID in relation Order. NULL not allowed, on delete/update cascade.
Store ID: foreign key refers to *Store ID* in relation Retail store. NULL not allowed, on delete/update cascade.

Places(*Order ID, Customer ID*)

Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.
Order ID: foreign key refers to order ID in relation Order. NULL not allowed, on delete/update cascade.

Retail(<u>StoreID</u>, Store location)
Employee(<u>Emplyoee ID</u>, *warehouse ID,* Employee Name)

Warehouse ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.

Works For(*Employee ID, Store ID*)

StoreID: foreign key refers to storerID in relation Retail store. NULL not allowed, on delete/update cascade.
Employee ID: foreign key refers to *Employee* ID in relation *Employee*. NULL not allowed, on delete/update cascade.

Warehouse(<u>Warehouse ID</u>, Warehouse location)
Products(<u>ProductID</u>, *Warehouse ID , Inventory ID*, Product Name)

Warehouse ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.
Inventory ID: foreign key refers to Warehouse ID in relation Warehouse. NULL not allowed, on delete/update cascade.

Contains(*Product ID, Store ID)*

Store ID: foreign key refers to storerID in relation Retail store. NULL not allowed, on delete/update cascade.
Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on delete/update cascade.

Ships(*Product ID, Supplier ID)*

Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on delete/update cascade.
Supplier ID: foreign key refers to Supplier ID in relation Supplier. NULL not allowed, on delete/update cascade.

Total...
2786

Shipped To(*Product ID, customer ID)*

Product ID: foreign key refers to Product ID in relation Product. NULL not allowed, on delete/update cascade.
Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.

Makes (*Payment ID, Customer ID*)

Customer ID: foreign key refers to Customer ID in relation Customer. NULL not allowed, on delete/update cascade.
Payment ID: foreign key refers to Payment ID in relation Payment. NULL not allowed, on delete/update cascade.

Payment(Payment ID,Payment Type)
Receive (*Store ID, Payment ID*)

StoreID: foreign key refers to storerID in relation Retail store. NULL not allowed, on delete/update cascade.
Payment ID: foreign key refers to Payment ID in relation Payment. NULL not allowed, on delete/update cascade.

Inventory System(InventoryID)

Supplier(Supplier ID, Supplier Name)

IV. Implementation of Relation Model via MySQL and NoSQL

MySQL Implementation:

The database was created in MySQL Workbench and the following queries were performed:

Query 1: Retrieve the names of all customers from the database.

```
SELECT CustomerName
FROM Customer;
```

CustomerName
Alice Williams
Robert Smith
Charles Brown
Diana Miller
Edwin Wilson
Fiona Clark
George Adams
Hannah Lewis
Ian Martinez
Julia Lopez
Kevin Reed
Laura Garcia
Michael Turner
RRdXdvCSMb
Olivia King
Paul Scott

Query 2: Find the total quantity of all orders placed in the system.

```
SELECT SUM(Quantity) AS TotalQuantity
FROM RSOrder;
```

Query 3: Display the order details (OrderID and Quantity) along with the store location where the orders were processed.

```
SELECT RSOrder.OrderID, RSOrder.Quantity,
RetailStore.StoreLocation
FROM RSOrder
INNER JOIN Processes ON RSOrder.OrderID =
Processes.OrderID
INNER JOIN RetailStore ON Processes.StoreID =
RetailStore.StoreID;
```

OrderID	Quantity	StoreLocation
1	88	NULL
2	89	NULL
3	84	NULL
4	75	NULL
5	30	NULL
6	4	NULL
7	64	NULL
8	39	NULL
9	5	NULL
10	13	NULL
11	35	NULL
12	22	NULL
13	100	NULL
14	77	NULL
15	24	NULL

Query 4: List all product names and their respective warehouse locations. If a product is not associated with any warehouse, include it in the result with a NULL location.

```
SELECT Products.ProductName, Warehouse.WarehouseLocation
FROM Products
LEFT JOIN Warehouse ON Products.WarehouseID =
Warehouse.WarehouseID;
```

ProductName	WarehouseLocati...
Laptop	New York
Smartphone	Los Angeles
Tablet	Chicago
Headphones	Houston
Monitor	Phoenix
Keyboard	Philadelphia
Mouse	San Antonio
Speaker	San Diego
Webcam	Dallas
Printer	San Jose
Scanner	Austin
Router	Jacksonville

Query 5: Retrieve the names of customers who have placed at least one order with a quantity greater than 10.

```
SELECT CustomerName
FROM Customer
WHERE CustomerID IN (
   SELECT CustomerID
   FROM Places
   WHERE OrderID IN (
      SELECT OrderID
      FROM RSOrder
      WHERE Quantity > 10
   )
)
```

CustomerName
Alice Williams
Robert Smith
Charles Brown
Diana Miller
Edwin Wilson
George Adams
Hannah Lewis
Julia Lopez
Kevin Reed
Laura Garcia
Michael Turner
RRdXdvCSMb
Olivia King

Query 6: Retrieve all orders where the quantity is greater than the average quantity of all other orders.

```
SELECT OrderID, Quantity
FROM RSOrder R1
WHERE Quantity > (
   SELECT AVG(R2.Quantity)
   FROM RSOrder R2
   WHERE R2.OrderID != R1.OrderID
);
```

OrderID	Quantity
1	88
2	89
3	84
4	75
7	64
13	100
14	77
17	84
18	70
19	94
20	69

Result

```
{ "_id" : 1, "name" : "Laptop", "price" : 999.99, "category" : "Electronics", "stock" : 50 }
{ "_id" : 2, "name" : "Smartphone", "price" : 599.99, "category" : "Electronics", "stock" : 100
{ "_id" : 3, "name" : "Headphones", "price" : 149.99, "category" : "Electronics", "stock" : 200
```

Query 7: Identify the store with the maximum number of processes among all stores that have a location starting with "Location-1."

```
GROUP BY StoreID
HAVING COUNT(*) >= ALL (
   SELECT COUNT(*)
   FROM Processes p
   JOIN RetailStore rs ON p.StoreID = rs.StoreID
   WHERE rs.StoreLocation LIKE 'Location-1%
   GROUP BY p.StoreID
);
```

StoreID
3
4
5
6
7
8
9
10

Query 8: Combine the locations of warehouses and retail stores into a single list, specifying their type as either "Warehouse" or "Store."

```
SELECT WarehouseLocation as Location, 'Warehouse' as Type
FROM Warehouse
UNION
SELECT StoreLocation, 'Store'
FROM RetailStore;
```

Location	Type
Mesa	Warehouse
Atlanta	Warehouse
Omaha	Warehouse
Colorado...	Warehouse
Raleigh	Warehouse
Miami	Warehouse
Virginia Be...	Warehouse
Oakland	Warehouse
Minneapolis	Warehouse
Tulsa	Warehouse
Arlington	Warehouse
Tampa	Warehouse
New Orleans	Warehouse
Wichita	Warehouse
Cleveland	Warehouse
Brooklyn, NY	Store
Santa Mon...	Store
Oak Park, IL	Store
Sugar Lan...	Store
Scottsdale...	Store

Query 9: For each warehouse, find its location and the number of products stored in it.

```
SELECT w.WarehouseLocation,
      (SELECT COUNT(*)
       FROM Products p
       WHERE p.WarehouseID = w.WarehouseID) as ProductCount
FROM Warehouse w;
```

WarehouseLocati...	ProductCount
New York	2
Los Angeles	2
Chicago	2
Houston	2
Phoenix	2
Philadelphia	2
San Antonio	2
San Diego	2
Dallas	2
San Jose	2
Austin	2
Jacksonville	2
Fort Worth	2
Columbus	2

Query 10: Calculate the average number of payments made per customer.

```
SELECT AVG(PaymentCount) as AvgPaymentsPerCustomer
FROM (
   SELECT CustomerID, COUNT(*) as PaymentCount
   FROM Makes
   GROUP BY CustomerID
) as CustomerPayments;
```

AvgPaymentsPerCustom...
1.0000

Implementation of the relational model in NoSQL:

The database was created in playground under Wine (Mongo) and the following queries were performed:

1. SIMPLE QUERY:

db.Product.find({ category: "Electronics" })

2. MORE COMPLEX QUERY:

```
db.Order.find({
  totalAmount: { $gt: 500 },
  status: { $in: ["Completed", "Shipped"] }
})
```

Result

```
{ "_id" : 1, "customerId" : 1, "products" : [ { "productId" : 1, "quantity" : 1 }, { "productId
```

3. AGGREGATE QUERY:

```
db.Order.aggregate([
  {
    $group: {
      _id: "$customerId",
      averageOrderTotal: { $avg: "$totalAmount" },
      totalOrders: { $sum: 1 }
    }
  },
  {
    $lookup: {
      from: "Customer",
      localField: "_id",
      foreignField: "_id",
      as: "customerInfo"
    }
  },
  {
    $project: {
      _id: 1,
      customerName: { $arrayElemAt: ["$customerInfo.name", 0] },
      averageOrderTotal: 1,
      totalOrders: 1
    }
  }
])
```

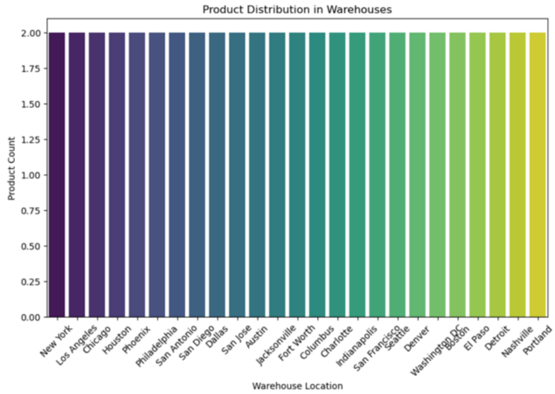
Result

```
{ "_id" : 3, "averageOrderTotal" : 119.98, "totalOrders" : 1, "customerName" : "Bob Johnson" }
{ "_id" : 2, "averageOrderTotal" : 659.96, "totalOrders" : 1, "customerName" : "Jane Smith" }
{ "_id" : 1, "averageOrderTotal" : 1299.97, "totalOrders" : 1, "customerName" : "John Doe" }
```

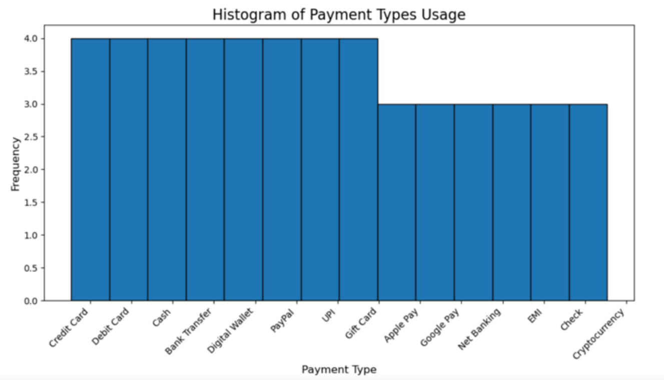
V. Database Access via Python

The database is accessed using Python and visualization of analyzed data is shown below. The connection of MySQL to Python is done using mysql.connector, followed by cursor.excecute to run and fetch all from query, followed by converting the list into a data frame using pandas library and using matplotlib to plot the graphs for the analytics.

Graph 1: Order Distribution buy Customer **Graph 2: Product Distribution in Warehouses**



Graph 3: Histogram of Payment Types Usage



VI. Summary and Recommendation

The Retail Store Inventory Management System is a complete answer to the many problems that contemporary retail operations encounter. The solution improves supply chain efficiency and inventory operations by incorporating cutting-edge technology including automation, real-time data collection, and sophisticated analytics. To minimize the risks of overstocking and stockouts and guarantee that the appropriate products are available to satisfy client demands, it precisely tracks inventory levels, orders, sales, and deliveries. The system's capacity to examine important variables, such as supply chain logistics, sales velocity, and purchase trends, permits a balanced stock flow and lowers human error in inventory control. Additionally, its real-time stock movement updates and automated reordering procedure enable shop managers to make well-informed judgments based on precise, current data.

Retail organizations are advised to adopt a phased rollout strategy to optimize the advantages of the Retail Store Inventory Management System. To enable system customization and fine-tuning based on feedback from the actual world, this should start with a pilot program in a few chosen stores. To guarantee that every employee is adept at utilizing the new system, staff training initiatives should be created and put into place. To find areas for development and make sure the system keeps up with the changing needs of the company, regular system audits and performance assessments should be carried out. Establishing a specialized team to oversee system performance, evaluate data insights, and formulate strategic suggestions based on the system's outputs is also advised.