SOFTWARE DESIGN DOCUMENT

GITHUB LINK: https//github.com/helen-liz/bread-supply-chain-system

G-18

SUPERVISOR NAME- DR. MARY NSABAGWA

STUDENT'S NAME	REGISTRATION NUMBER	STUDENT NUMBER
AMARA ROY	24/U/03334/EVE	2400703334
MWANABASA HANES	24/U/22867	2400722867
AKELLO HELLEN ELIZABETH	24/U/03152/EVE	2400703152
OPETO ISAAC	24/U/10762/PS	2400710762
ARMAND SHEMATSI	23/X/25074/PS	2300725074

TABLE OF CONTENTS

Contents

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Purpose

The purpose of this design document is to outline the architecture functionality, and implementation of bread supply chain management system.

The system is designed to optimize the entire supply chain process, from raw material sourcing to the final project distribution, ensuring efficiency, quality and cost-effectiveness. By integrating data analytics, order tracing and inventory management, the system will enhance decision making, reduce waste, and improve supply chain transparency.

The system will facilitate procurement, production, inventory management, distribution, and order processing, ensuring smooth operations and minimizing waste. By leveraging data-driven insights and automation, this system will enhance decision making and optimize resource allocation.

1.2 Scope

The system covers all stages of bread production and distribution, including:

- Raw material procurement (flour, yeast, preservatives, cooking oil among others).
- Production process management to ensure efficiency and quality control.
- Inventory tracking to monitor stock levels across manufacturing units and distribution centers.
- Order processing for retailers, wholesalers, and end consumers.
- Supply chain optimization using machine learning predictions and demand forecasting.
- Order fulfillment – Efficient processing of orders from retailers, wholesalers, and consumers.
- Logistic coordination – Tracking deliveries and optimizing distribution routes.
- Stakeholder collaboration via a built-in communication tool for suppliers, distributors, and retailers.
- Vendor verification to ensure compliance with regulatory and financial standards

The primary goals of the system are to:

- Provide a seamless order placement and processing experience for customers and bakery staff.
- Ensure product quality and consistency by accurately managing ingredient quantities and baking schedules.
- Automate workflow management, including inventory tracking, supply restocking, and order fulfillment.
- Generate detailed performance and financial reports for bakery operations. Provide analytics to identify trends in customer preferences and operational efficiencies

1.3 Overview of the Document

This document is systematically organized into the following sections:

Chapter 1: Establishes the foundation of the document, outlining its purpose, scope, key references, and definitions of industry-specific terminology.

Chapter 2: Provides an overview of the Bimbo Bakery Operations System, detailing its core functionalities, operational context, and structural design.

Chapter 3: Defines the system architecture, illustrating the design framework, its modular decomposition, and the rationale behind key architectural decisions.

Chapter 4: Focuses on data management, showcasing how essential bakery-related information—such as ingredient tracking, inventory levels, and order processing—is structured within the system.

Chapter 5: Describes system components, breaking down individual modules responsible for bakery production workflows, order handling, and business analytics.

Chapter 6: Explores the human interface design, demonstrating how bakery personnel interact with the system, complemented by interface visuals for enhanced clarity.

1.4 Reference Material

https://drive.google.com/file/d/1stfsdTL-5yoho-jlzpxVGMKcxdxSEuaU/view?usp=sharing

1.5 Definitions and Acronyms

1.51 Acronyms

Table 1. 1 Acronyms and descriptions

Acronym	Description
SDD	**Software Design Document**
CLI	**Command Line Interface**
ERD	**Entity Relationship Diagram**
PHP	**Hypertext Preprocessor**
DFD	**Data flow Diagram**

1.52 Definitions

Software Design Document; A structured document outlining the blueprint of software, assisting in analysis, strategic planning, implementation, and informed decision-making during the development lifecycle**.**

Use Case Diagram; A visual representation summarizing system functionalities, highlighting interactions between various user roles (actors) and the system itself.

Entity Relationship Diagram (ERD); A conceptual framework that visually maps out relationships between database entities, serving as a foundation for structured data organization.

Data Flow Diagram (DFD); A diagram that illustrates how data moves within the system, illustrating processes, data sources, and pathways for efficient information flow**.**

Hypertext Preprocessor (PHP); A widely used open-source scripting language embedded within HTML, enabling

2. System overview

The Bread Supply Chain Management System provides a comprehensive platform to monitor and manage the end-to-end process of bread production — from sourcing raw materials (for example, flour, yeast, salt) to delivering finished loaves to retail outlets. The system aims to optimize the supply chain using real-time data, predictive analytics, and machine learning.

2.1 Key Features and Functions

Demand Forecasting

Demand Forecasting: Predict future bread demand based on historical sales, seasonal trends, and regional consumption patterns.

The system looks at past records like how bread is sold every day or week and uses that information to guess the future demand for example if people buy bread more on Friday then the system will learn to bake more on Fridays. This is done using the machine learning subsystem that takes in historical data, finds patterns and makes predictions. So laravel sends the sales data to machine learning subsystem and it sends back the forecast. Then laravel shows this prediction back to the user so they can plan better on baking right amount of bread, buying the ingredient and avoiding waste.

Customer Segmentation

Identify customer segments based on their buying behavior (e.g., frequency, type of bread preferred) to personalize inventory and marketing strategies.

The system looks at customer data like how often each person buys bread, what types of bread they like, how much they spend and when they usually make purchases. Using this information, the system groups similar customers together for example it might create a group of regular customers who buy every week, another group that buys during holidays and another for customers who buy specific type of bread using machine learning. Laravel ends customer data to the machine learning subsystem which runs segmentation and sends back the results. Laravel then shows this groups in way that will help the system treat each group

Core functional modules

Chat function

Enable communication between wheat/flour suppliers, bakeries, and retailers to resolve issues like delays, shortages, or quality concerns.

Each user logs into the system through their account using laravel, the system identifies the user type for example supplier, customer so the user opens the chat and they can select the person they want to message. The user sends the message and it is stored in the database

The chat function can also be used for customer support.

Order Processing

A user logs into the system, they go to place order. The system receives the order request and stores in the MySQL database, a confirmation message is sent to the user, the system notifies the staff that a new order has arrived.

The system checks inventory then assigns a delivery team, real time tracking can be shown to the customer. Once bread is delivered, order is closed.

Inventory management

The system keeps records of how much raw material and finished bread is available, every time new raw materials arrive stock is updated and when bread is baked or orders are made the changes happen automatically. When the bakery prepares bread, it subtracts the used ingredients.

The system checks if any material is running low, it sends an alert, it also generates automated reports.

Inventory is checked before order processing, demand forecasting and vendor validation.

Workforce distribution management

Each worker is registered in laravel subsystem with a specific role, their availability and working hours as well are stored in the MySQL database. The system allows the supervisor to assign tasks to each worker. It also creates work schedules and assigns shifts across the work force .

Workers log in to see their assigned tasks and update progress for example task completed.

If someone is absent, the system reassigns available staff and more workers are assigned when more orders come.

Scheduling tasks

The system knows different stakeholders, on a schedule, it collects relevant data from the database and uses predefined templates to create reports and tailors them to each stakeholder.

The system sends or displays reports and these tasks repeat automatically.

Vendor validation

The vendor submits application containing financial information, business reputation then laravel uploads and sends data to the java server via an API call. The java server performs validation then if the validation is successful the vendor marks as pre-approved and if it fails it shows an error message with reason.

It schedules the physical visit automatically.

Analytics dashboard

The laravel backend constantly pulls updated data from the MySQL database, laravel processes raw data into meaningful summaries then the dashboard displays information using bar charts, tables

3. Software Architecture

3.1 Architectural Design

Bimbo Bread Supply Chain Management System follows a modular client server architecture that organizes each with specific responsibilities. The modular program structure enhances separation of concerns, reusability, maintainability and scalability.

The system is divided into the following high level sub-systems.

Client Layer

Machine Learning

Server layer

Client layer (Laravel-user interface)

Provides user interface for customers, managers, vendors and staff

It handles bread orders, inventory tacking, scheduling, chat and recommendations.

It sends requests to the server via API calls

Machine Learning Subsystem

Implements predicting future bread demand, segment customers based on purchasing behavior.

It implements customer segmentation by classifying customers in groups according to buying behavior.

Server Layer (java and MySQL):

Java server

Performs validation checks for vendors registering into the system

It processes requests from Laravel.

Database (MySQL)

Stores supply chain data, vendor details, inventory levels and transaction record.

Collaboration between the subsystems

Users place orders, track inventory and communicate through Laravel based user interface, laravel sends requests to the backend API for order processing, inventory updates and vendor authentication.

The java server receives user requests from laravel and manages vendor authentication, order validation and inventory updates, it processes the PDF applications for vendor validation and sends the processed data to the MySQL database for storage.

MySQL stores vendor details, transactions, inventory levels, and historical order data then it provides data to the web interface upon request to ensure real-time tracking then feeds the structured data into the machine learning module for analysis

Machine learning System uses historical data from MySQL to predict future demand trends, segment customers based on purchasing behavior to optimize distribution and sends results back to the java server

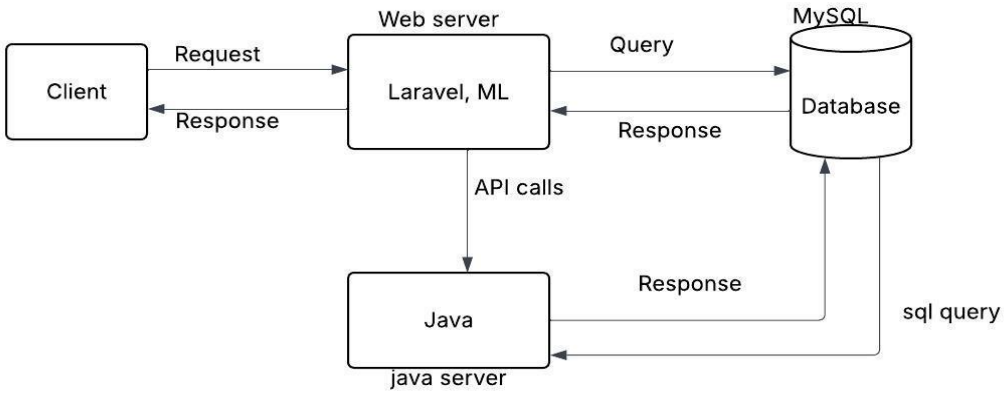

Figure 3. 1 Architectural diagram

Figure 3.1 shows the major subsystems and data repositories and their interconnections. The client makes a request to the webserver and the webserver sends back the response to the client, the webserver communicates to the MySQL database and the database communicates with the java server.

3.2 Decomposition Description

Client Layer (laravel)

It acts as the user interface and central controller for all the business functions. Manages user interaction, data entry, report viewing, and communication between the sub-systems

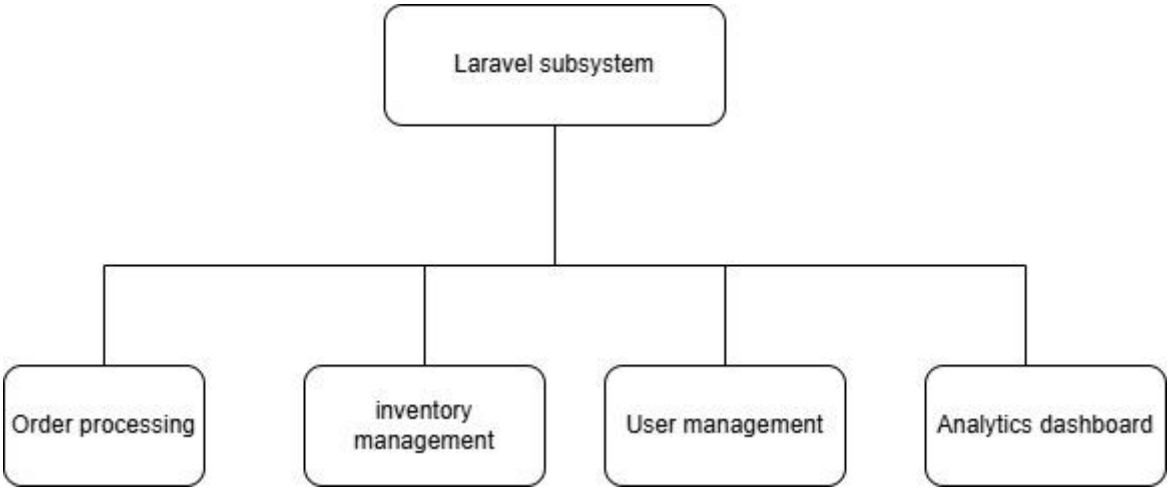

Figure 3. 2 Tree structure for the laravel subsystem

Figure 3.2 shows the major modules that will be built in the laravel sub system

The Order Processing Module manages bread orders, cart handling, payment integration and order status update.

The Inventory Management Module tracks bread stock levels for example white, brown, buns, updates stock after sales and checks for re-orders

User management module allows users to sign up and the existing users to login using secure authentication

Analytics dashboard module displays analytics from machine learning sub-system

Machine Learning Subsystem

It is an external service for performing analysis and predictions, segments customers based on past purchase patterns to guide stock and marketing strategies

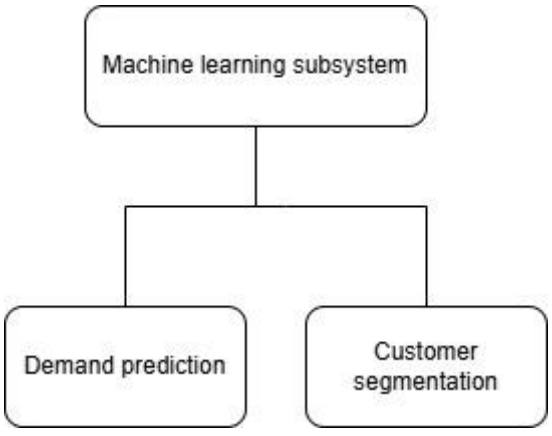

Figure 3. 3 Decomposition of machine learning subsystem

Figure 3.3 shows the major functions in the machine learning subsystem

Demand Prediction Module trains and predicts future bread demand from sales data

Customer Segmentation Module groups customers using clustering based on buying patterns

Java server

Validates vendors registering in the system to ensure legitimacy and approval status

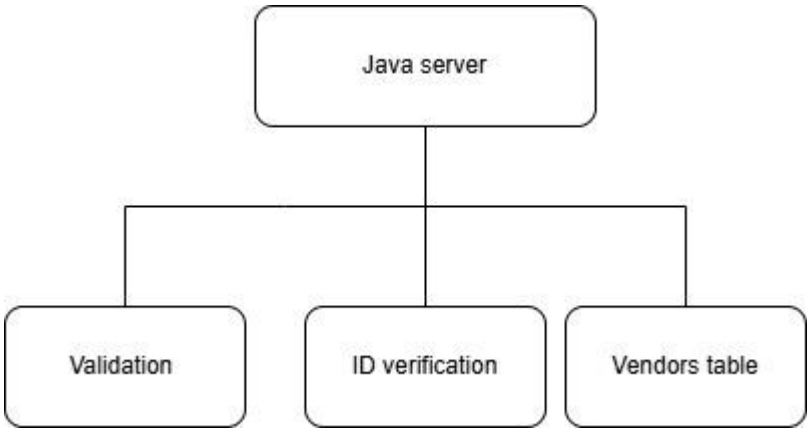

Figure 3. 4 Components of the java server

Figure 3.4 shows the major functions in the java server

Validation API Controller receives POST requests with vendor data

ID Verification Module checks ID formats or fetches data from national registry

Response Builder Module returns a validation report back to Laravel

Database Layer

Stores all persistent data- orders, users, inventory, vendor profiles and analytics results

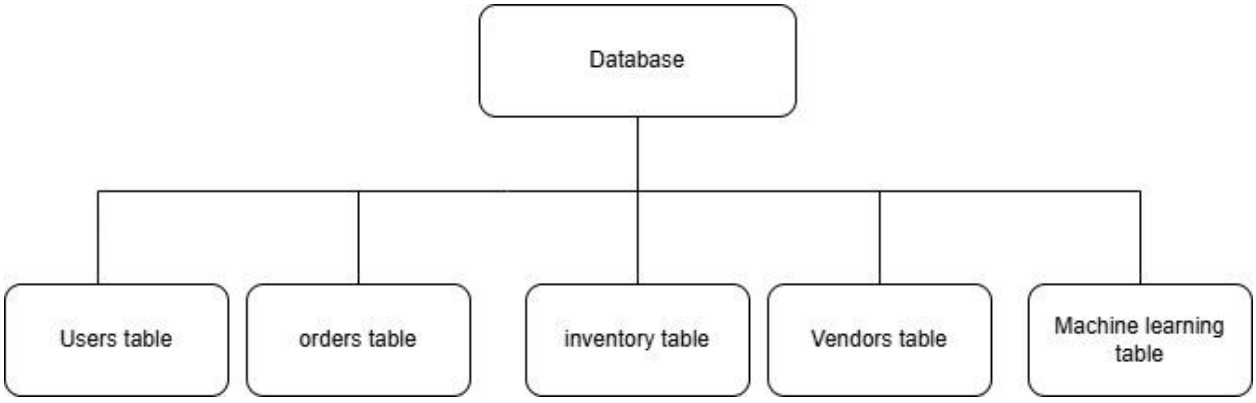

Figure 3. 5 Tree structure for the database

Figure 3.5 shows the tables that the database layer contains

Users table stores all user records (Admins, Vendors, Staff)

Orders table stores order data including bread type, quantity and status.

Inventory table stores stock levels, batch IDs, expiry dates

Vendors table stores vendor credentials, validation status

Machine learning predictions table stores forecast results, segmentation labels

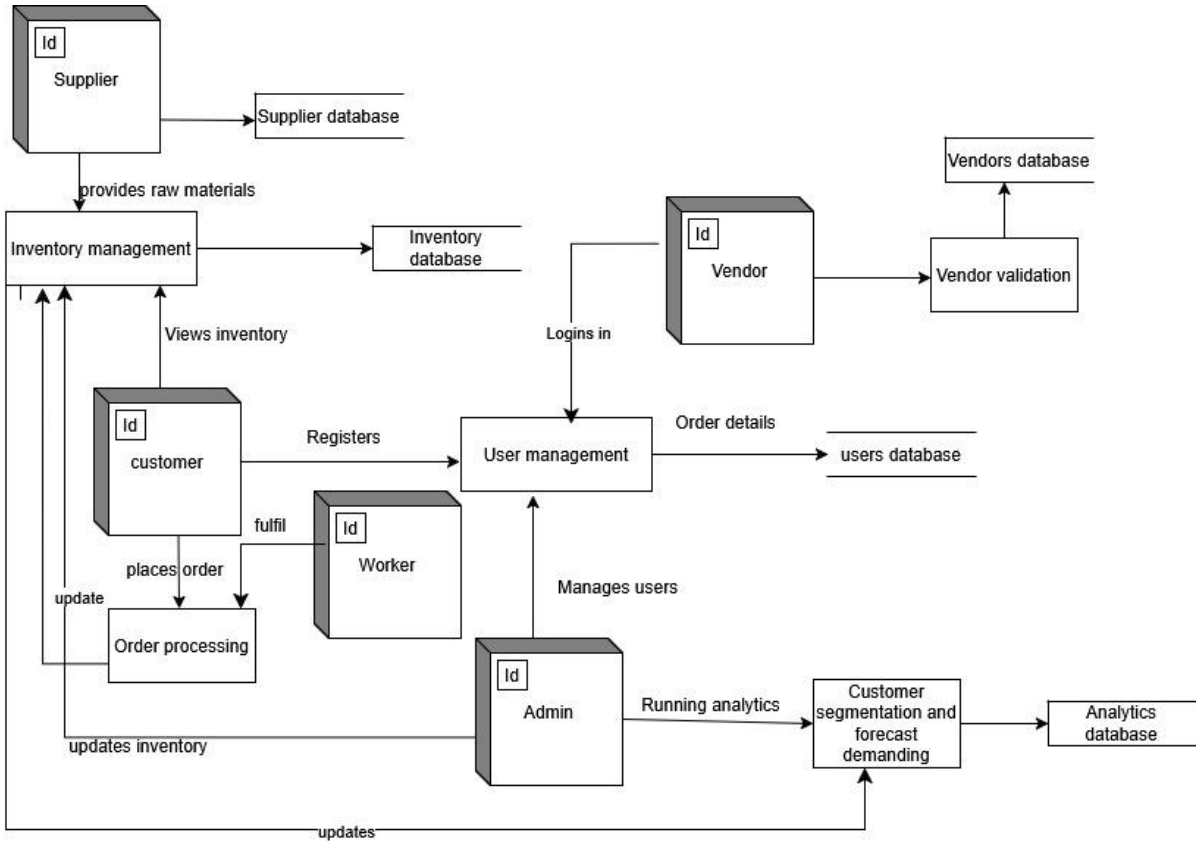

Figure 3. 6 Data flow diagram

Figure 3.6 shows how data flows in the bread supply chain management system and the external entities

3.3 Design Rationale

The client-server architecture was chosen as the foundational design for the Bread Supply Chain Management System due to its modularity, clear separation of concerns, and scalability. This approach aligns well with the complexity of managing bread production, distribution, sales, and analytics within an interconnected system that serves multiple user roles — including vendors, managers, inventory officers, and administrators.

Critical Issues Considered

Modularity and Separation of Concerns

Client-server allows each component (Laravel, Java, ML, MySQL) to be independently developed and maintained.

Scalability

The system must support multiple clients placing orders or making vendor requests at once.

Client-server supports horizontal scaling — add more server power or load balancers as traffic grows.

Security

Sensitive operations (e.g., vendor verification, customer data, analytics) are done on secure servers.

Client-server architecture allows centralized control of access, input validation, and data privacy.

Maintainability and Upgradability

Each subsystem can evolve independently, reducing risk and improving maintainability.

Reliability and Consistency

Data and business logic are stored centrally, ensuring consistent behavior across all clients.

Clients rely on the server to provide the latest validated and predicted information. Testability

Each subsystem (client, ML API, Java server) can be unit tested and debugged separately.

PROS

Centralized Control

All data and logic are stored and managed in one place (the server).

Easy Maintenance

Updates, bug fixes, or changes can be done on the server without touching clients.

Scalability

You can handle more clients by upgrading the server or using load balancers.

Security

Sensitive operations are done server-side, reducing risk of client-side tampering.

CONS

Server Dependency

If the server goes down, all clients are affected — there's a single point of failure.

Higher Server Load

All processing and data handling is done on the server, which can become overloaded.

Complex Setup

You must configure multiple technologies (Laravel, Java, ML, MySQL) and handle their integration.

Network Reliance

Requires constant internet, clients can't function offline.

4. Data design

4.1 Data description

In the Bread Supply Chain Management System, the information domain encompasses a variety of real-world entities and activities including customers, suppliers, orders, inventory, deliveries, sales records, work force schedules, and machine learning outputs.

This raw information is transformed into structured digital representations through well-designed relational database schemas and integrated APIs between Laravel, Java, and the machine learning module.

The system uses MySQL as the primary database, with Laravel's Eloquent ORM responsible for mapping real-world concepts into data structures such as tables, columns, rows, and relationships. Eloquent handles the creation, reading, updating, and deletion of records based on defined models (e.g., Product, Customer, Order, Vendor, Inventory, among others).

Key transformations include:

Customer Data: Information like name, contact, and purchase history is captured through forms on the Laravel frontend and stored in the customers table.

Sales Data: This is stored in a sales table and used by the machine learning module to predict future demand and perform customer segmentation. Laravel transfers this data to the ML module through a REST API, and predictions are returned and stored in a ml_forecasts table.

Order Processing: When orders are placed, Laravel stores them in the orders table, linked to customers and bread products via foreign keys.

Vendor Validation: PDF applications submitted by prospective vendors are uploaded through Laravel, forwarded to the Java server (via HTTP requests), and validation results are returned and stored in a vendor_validations table.

Inventory Management: Inventory levels are tracked using an inventory table, which records current stock, reorder levels, and alerts.

Workforce Distribution: Staff shift data is structured into a workforce_schedule table to track availability and assignment across supply points.

Scheduled Reporting: Laravel's scheduler triggers report generation based on data summaries, which are stored in report_logs or exported for stakeholders.

Chat Function: Laravel stores messages in a messages table, linked to sender and receiver IDs, allowing asynchronous communication between suppliers and consumers.

Data is processed and accessed using SQL queries abstracted through Laravel's query builder and models. Each table is normalized to reduce redundancy, and foreign keys enforce referential integrity between related entities. The overall architecture ensures data flows efficiently across the Client-Server structure, from user interfaces (browsers), through the Laravel controller logic, to either the ML subsystem or the Java server as needed, and back into the database.

User's table

The User table stores essential information about individuals who interact with the system, including customers, administrators, suppliers and all staff members.

It maintains unique identifiers for each user, alongside their personal details like name, contact information, and assigned roles within the system.

This table helps facilitate authentication, authorization, and personalized interactions based on user type, ensuring efficient management of access and activities.

Table 4. 1 Users table

Column Name	Data Type	Description
user_id	INT	Unique identifier for each user
username	VARCHAR	Username for login
password	VARCHAR	Securely hashed password
role	VARCHAR	User role (supplier, factory, wholesaler, retailer)
email	VARCHAR	Email address for communication
phone	VARCHAR	Contact phone number

Machine Learning Data

Machine learning data in a bread supply chain system consists of structured datasets used to train predictive models for optimizing production, inventory, sales, and logistics.

This data includes historical sales records to forecast demand, customer purchasing behavior for personalization, production efficiency metrics to predict bottlenecks, and logistics performance data for route optimization.

ENTITY RELATIONSHIP DIAGRAM

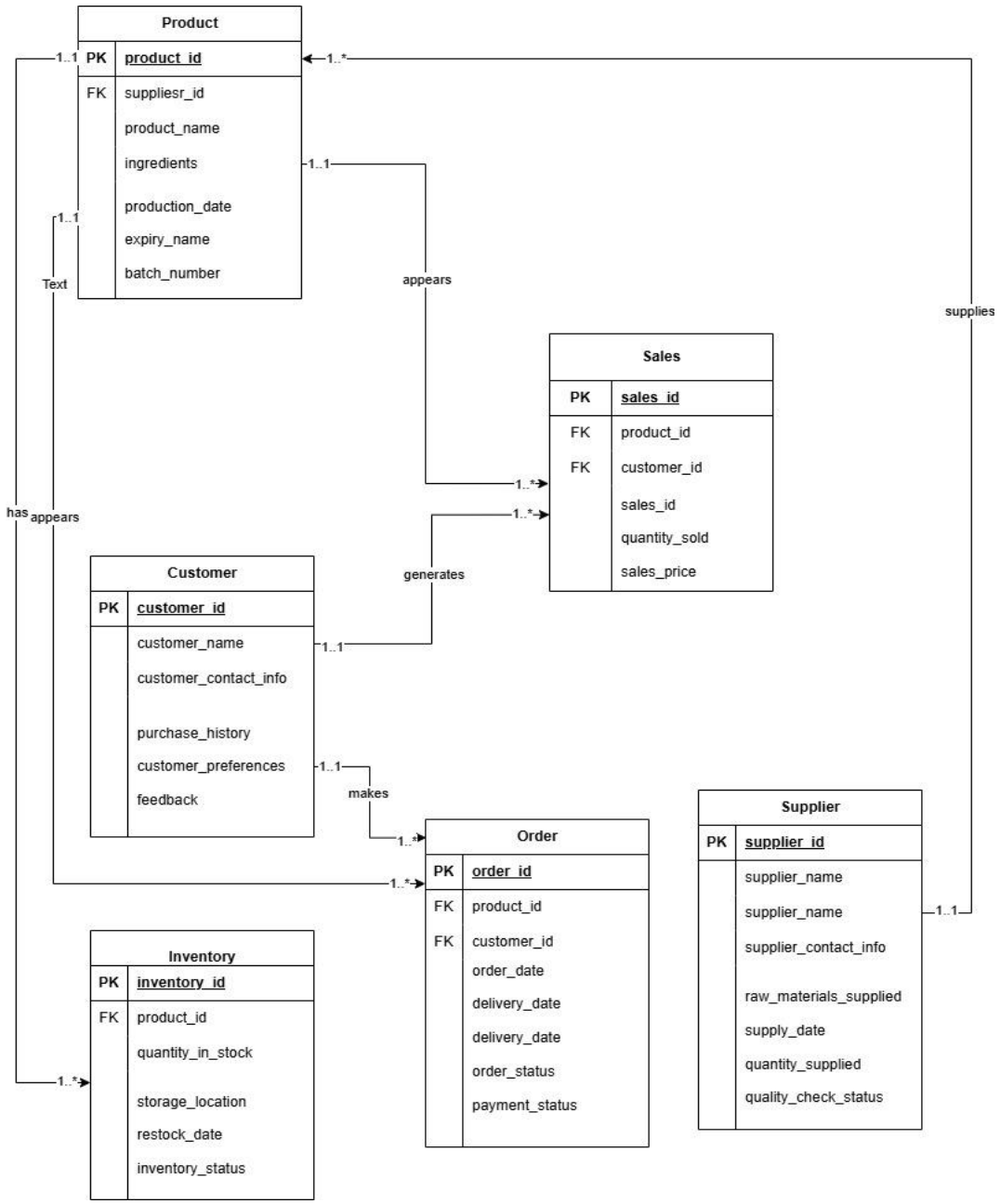

Figure 4. 1 Entity relationship diagram

DESCRIPTION

Product and Sales

In the Sales table, you have a foreign key(product_id) that links each sales record to a specific product in the Product table.This shows that each sale is associated with one product.

One-to-Many -A single product can appear in many sales records because it can be sold maultiple times.

Product and Inventory

The inventory table includes a foreign key(product_id), which means that every inventory entry is associated with a product. This helps you track how many units of a particular product are available, where they are stored, and when you need to restock.

Relationship Type:

One-to-Many: If one product can be stored in multiple locations or batches(for example, different warehouses or different storage conditions), then one product might have severalinventory records.

Product and Orders

The order table also uses product_id as a foreign key, indicating that each order is placed for a specific product. Similarly, the Order table includes a customer_id foreign key to record which customer made the order.

Relationship Type:

One-to-many:

A product can be ordered many times, meaning one product can appear in multiple records.

Similarly, one customer can place many orders, but each order is linked to one specific customer and one specific product.

Customer with Sales and Orders

Customer table is connected to both the Sales and Order tables via a foreign key (customer_id).This serves as a record of who is purchasing or ordering products.

Relationship ship Type:

One-to-Many:

A customer may have multiple sales transactions, meaning they can multiple products over time.

A customer can also have multiple records.

Supplier and Product

The Product has supplier_id foreign key indicating each product is associated with a supplier, meaning the supplier is responsible for supplying that product to the business.

Relationship Type:

Assumption:

If one product is sourced from only one supplier, this foreign creates a **one-to-many** relationship (one supplier can supply many product.

4.2 Data dictionaries

A **data dictionary** is a structured document that defines the attributes, data types, relationships, and constraints of a database.

It provides clarity on the meaning, purpose, and format of each field, ensuring consistency in data management.

By detailing elements like primary keys, foreign keys, and validation rules, it helps developers, analysts, and administrators maintain database integrity and optimize system functionality.

Essentially, it acts as a reference guide for understanding and organizing the database schema.

Our data dictionary for Bimbo bread supply system encompasses products, suppliers, customers, sales, inventory and orders.

Table 4. 2 Product data dictionary

Attribute	**Data type**	**Description**
Product_ID	VARCHAR(25)	Unique identifier for each bread type
Product_name	VARCHAR(100)	Name of bread
Ingredients	VARCHAR(100)	List of ingredients used
Production_date	DATE	Date when the bread was baked
Expiry_date	DATE	Shelf life duration
Batch_number	INT	Unique identifier for each batch

Table 4. 3 Supplier data dictionary

Attribute	**Data type**	**Description**
Supplier_ID	VARCHAR(25)	Unique identifier for suppliers
Supplier_name	VARCHAR(100)	Supplier name
Supplier_contact_info	VARCHAR(150)	Phone number, email
Raw_materials_supplied	VARCHAR(200)	Type of raw materials delivered
Supply_Date	DATE	Date materials where supplied.
Quantity_supplied	INT	Number of raw materials delivered
Quality_check_status	ENUM	Pass/Fail status after inspection

Table 4. 4 Customer data dictionary

Attribute	Data type	Description
Customer_ID	VARCHAR(25)	Unique identifier for each customer
Customer_name	VARCHAR(100)	Name of the customer
Customer_contact_info	VARCHAR(150)	Phone/Email details
Purchase_history	VARCHAR(100)	Record of the previous transaction
Customer_preferences	VARCHAR(100)	Favorite bread type
Feedback	ENUM	Ratings and reviews left by customers

Table 4. 5 Sales data dictionary

Attributes	**Data type**	**Description**
Sales_ID	VARCHAR(50)	Unique identifier for the sales transactions
Product_ID	VARCHAR(50)	Identifier for the bread sold
Customer_ID	VARCHAR(50)	Identifier for the customer purchasing
Sales_date	DATE	Date of purchase
Quantity_sold	INT	Number of bread unit sold
Sales_price	CURRENCY	Final cost per unit
Payment_method	ENUM	Cash, card, digital payment

Table 4. 6 Inventory data dictionary

Attribute	**Data type**	**Description**
Inventory_ID	VARCHAR(50)	Unique identifier for inventory tracking
Product_ID	VARCHAR(50)	Identifier for stored bread
Quantity_in_stock	INT	Amount available
Storage_location	VARCHAR(50)	Where the bread is stored
Restock_date	DATE	Date new stock should be added
Inventory_status	ENUM	Availability (In Stock, Low Stock, Out of Stock)

Table 4. 7 Order data dictionary

Attribute	**Data type**	**Description**
Order_ID	VARCHAR(50)	Unique identifier for each customer order
Product_ID	VARCHAR(50)	Identifier for ordered bread
Customer_ID	VARCHAR(50)	Identifier for the ordering customer
Order_date	DATE	Date the order was placed
Delivery_date	ADATE	Expected delivery schedule
Order_status	ENUM	Processing, Delivered, Cancelled
Payment_status	ENUM	Paid, Pending, Failed

5. Component Design

The system consists of the following components:

- **Order Processing** -Manages bread orders, cart handling, payment integration and order status update.
- **Inventory Management**-Tracks bread stock levels for example white, brown, buns, updates stock after sales and checks for re-orders
- **User management** -Allows users to sign up and the existing users to login using secure authentication
- **Reports & Analytics**-Displays analytics from machine learning sub-system
- **Demand Prediction** -Trains and predicts future bread demand from sales data
- **Customer Segmentation** -It groups customers using clustering based on buying patterns.
- **Vendor Registration**- Validates vendor details to ensure data quality and compliance.
- **Request Processing**-Receives and processes requests from the Laravel client, enforcing business rules.
- **Supply Chain Data Management-** Stores all details related to supply chain operations, product flows, among others.
- **Vendor Details**- Maintains a secure record of vendor profiles and registration data.
- **Inventory Levels**- Tracks current inventory, supporting real-time data on stock availability.
- **Transaction Records-** Logs all transactions, orders, and payment data.

Component-Level Algorithm Summaries (Pseudocode)

Order Processing

Manages bread orders, cart handling, payment integration and order status update.

pseudocode

```
FUNCTION ProcessOrder(customerID, productID, quantity):
IF CheckStock(productID, quantity) == TRUE:
orderID = GenerateOrderID()
UPDATE OrderStatus(orderID, "Processing")
 SET DeliverySchedule(orderID)
RETURN OrderConfirmation
```

ELSE
RETURN "Insufficient Stock"

Inventory Management

Tracks bread stock levels for example white, brown, buns, updates stock after sales and checks for re-orders

pseudocode

FUNCTION UpdateInventory(productID, quantityUsed):
currentStock = GetStock(productID)
newStock = currentStock – quantityUse
UPDATE Stock(productID, newStock)
IF newStock < Threshold:
TriggerRestock(productID)
RETURN InventoryStatus

User Management

Allows users to sign up and the existing users to login using secure authentication

pseudocode

FUNCTION RegisterUser(userType, name, contact):
IF userType == "Customer":
CREATE CustomerProfile(name, contact)
ELSE IF userType == "Supplier":
CREATE SupplierProfile(name, contact)
ELSE:
CREATE AdminProfile(name, contact)
RETURN UserID

Reports & Analytics

Displays analytics from machine learning sub-system

pseudocode

FUNCTION GenerateReport(reportType
IF reportType == "Sales"
RETURN FetchSalesData()
ELSE IF reportType == "Inventory":

RETURN FetchInventoryStatus()
ELSE IF reportType == "Supplier":
RETURN SupplierPerformanceReport()
RETURN "Invalid Report Type"

Demand Prediction

Trains and predicts future bread demand from sales data.

pseudocode

function predict_demand(historical_sales, seasonal_factors, economic_trends):

weighted_sales = apply_weights(historical_sales, seasonal_factors, economic_trends)

predicted_demand = statistical_model(weighted_sales)

return predicted_demand

Customer Segmentation

It groups customers using clustering based on buying patterns.

pseudocode

function segment_customers(customer_data):

clusters = apply_clustering_algorithm(customer_data)

categorized_customers = assign_labels(clusters)

return categorized_customers

Vendor Registration

It validates vendor details to ensure data quality and compliance.

Pseudocode

FUNCTION ValidateVendorRegistration(vendorData)

IF vendorData.name IS NULL THEN RETURN Error("Vendor name is required")

IF vendorData.email IS NULL THEN RETURN Error("Vendor email is required")

IF vendorData.licenseNumber IS NULL THEN RETURN Error("License number is required")

IF NOT IsValidEmail(vendorData.email) THEN RETURN Error("Invalid email format")

IF NOT IsValidLicense(vendorData.licenseNumber) THEN RETURN Error("Invalid license number")

IF vendorData.documents IS NOT NULL THEN

FOR EACH document IN vendorData.documents DO

IF NOT IsValidDocument(document) THEN RETURN Error("One or more documents are invalid")

END IF

END FOR

END IF

RETURN Success("Vendor registration validated successfully")

Request Processing

Receives and processes requests from the Laravel client, enforcing business rules.

Pseudocode

FUNCTION ProcessRequestFromLaravel(request)

requestType = request.type

SWITCH(requestType):

CASE "RegisterVendor"

validationResult = ValidateVendorRegistration(request.data)

IF validationResult IS Error THEN: SendResponseToLaravel(validationResult) ELSE

dbResult = SaveVendorToDatabase(request.data)

IF dbResult IS Error THEN: SendResponseToLaravel(Error("Database error when saving vendor"))

ELSE: SendResponseToLaravel(Success("Vendor registration successful"))

END IF

END IF

BREAK

CASE "GetInventory":

inventoryData = QueryInventoryLevels(request.parameters) SendResponseToLaravel(inventoryData)

BREAK

CASE "PlaceOrder"

orderResult = ProcessOrder(request.data)

IF orderResult IS Success THEN:

UpdateTransactionRecords(orderResult.orderID, request.data)
UpdateSupplyChainData(orderResult.orderDetails)

END IF SendResponseToLaravel(orderResult)

BREAK

DEFAULT: SendResponseToLaravel(Error("Unknown request type"))

Supply Chain Data management

Stores all details related to supply chain operations, product flows, among others.

pseudocode

FUNCTION UpdateSupplyChainData(orderDetails)

FOR EACH item IN orderDetails.items DO:

currentStock = QueryInventoryForItem(item.itemID)

newStock = currentStock - item.quantity

IF newStock < 0 THEN: RETURN Error("Insufficient stock for item " + item.itemID)

END IF UpdateInventory(item.itemID, newStock)

END FOR

log currentTime = GetCurrentTimestamp() SQL = "INSERT INTO SupplyChainLog (orderID, timestamp, details) VALUES (orderDetails.orderID, currentTime, orderDetails.summary)" EXECUTE SQL RETURN Success("Supply chain updated successfully")

Vendor Details

Maintains a secure record of vendor profiles and registration data.

pseudocode

FUNCTION SaveVendorToDatabase(vendorInfo)

currentDate = CURRENT_DATE SQL = "INSERT INTO VendorDetails (vendorID, name, email, licenseNumber, registrationDate, documents) VALUES (?, ?, ?, ?, ?, ?)" PARAMETERS = (vendorInfo.id, vendorInfo.name, vendorInfo.email, vendorInfo.licenseNumber, currentDate, vendorInfo.documents)

result = EXECUTE SQL WITH PARAMETERS

IF result indicates failure THEN: RETURN Error("Database error: could not save vendor data")

ELSE: RETURN Success("Vendor saved successfully")

Inventory Levels

Tracks current inventory, supporting real-time data on stock availability.

pseudocode

FUNCTION QueryInventoryForItem(itemID)

SQL = "SELECT stock FROM InventoryLevels WHERE itemID = ?"

PARAMETERS = (itemID)

result = EXECUTE SQL WITH PARAMETERS stockLevel = result.stock

RETURN stockLevel

UpdateInventory(itemID, newStock)

SQL = "UPDATE InventoryLevels SET stock = ? WHERE itemID = ?"

PARAMETERS = (newStock, itemID)

result = EXECUTE SQL WITH PARAMETERS

IF result indicates failure THEN:

RETURN Error("Failed to update inventory for item: " + itemID)

ELSE: RETURN Success("Inventory updated for item " + itemID)

Transaction Records

Logs all transactions, orders, and payment data.

pseudocode

FUNCTION ProcessOrder(orderData)

IF orderData.orderItems IS EMPTY THEN: RETURN Error("Order must contain at least one item")

orderID = GenerateUniqueOrderID()

currentTime = GetCurrentTimestamp()

FOR EACH orderItem IN orderData.orderItems DO: SQL = "INSERT INTO TransactionRecords (orderID, itemID, quantity, price, timestamp) VALUES (?, ?, ?, ?, ?)"

PARAMETERS = (orderID, orderItem.itemID, orderItem.quantity, orderItem.price, currentTime)

result = EXECUTE SQL WITH PARAMETERS IF result indicates failure THEN: RETURN Error("Failed to record order item: " + orderItem.itemID)

END IF

END FOR SQL = "INSERT INTO Orders (orderID, vendorID, totalAmount, timestamp) VALUES (?, ?, ?, ?)"

PARAMETERS = (orderID, orderData.vendorID, orderData.totalAmount, currentTime) result = EXECUTE SQL WITH PARAMETERS IF result indicates failure THEN: RETURN Error("Failed to record overall order details")

END IF

RETURN Success("Order processed successfully with Order ID: " + orderID, orderDetails=orderData)

Systemic Flow Overview

Each component works systematically:

- **Orders checked against inventory**. Payment is processed.
- **Production triggers stock updates** . Inventory adjusts levels.
- **Users register** . Customers place orders.
- **Reports summarize data** . Business insights are generated.
- If all checks pass, return a success message indicating that validation is successful.
- Return success when the supply chain is updated successfully.
- The vendor details are now stored securely in the database.
- Output the current inventory count.
- Send a success message along with the unique Order ID.
- If the request type is unrecognized, return an error response. The response is sent back to Laravel

6. Human Interface Design

6.1 Overview of User Interface

The Bimbo Bread Supply Chain System provides distinct interfaces for administrators, suppliers, bakery managers, distribution teams, and retail managers.

Administrators:

Access via web interface to manage vendors, view analytics, and configure system settings

Dashboard with KPIs for entire supply chain

Suppliers:

Raw material inventory tracking interface

Order request management system

Integrated chat with bakeries

Bakery Managers:

Production monitoring dashboard

Workforce scheduling tools

Machine maintenance alerts

Distribution Teams:

Route optimization interface

Vehicle tracking system

Delivery confirmation system

Retail Managers:

Bread ordering portal

Inventory level monitoring

Demand forecast displays

6.2 Screen Images

1. Administrator Login Screen

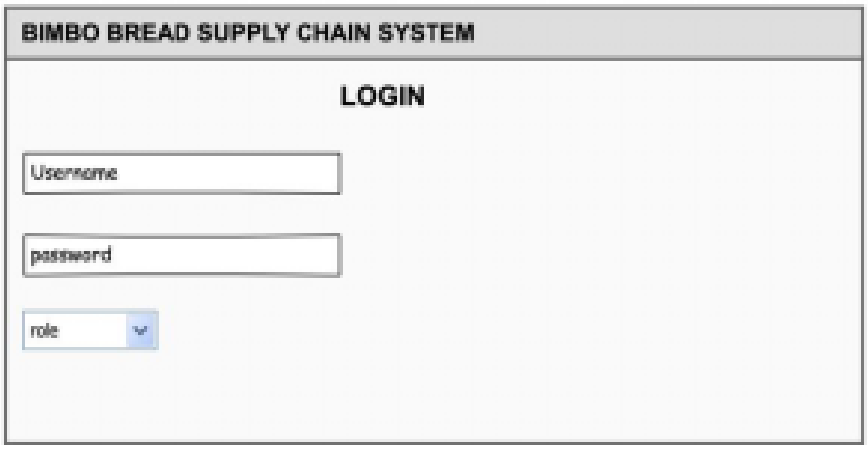

Figure 6. 1 Administrative login screen image

Figure 6.1 shows the login for system administrators, featuring fields for username and password, along with a login button for authentication. The design emphasizes security and ease of access for authorized personnel.

2. Supplier Dashboard

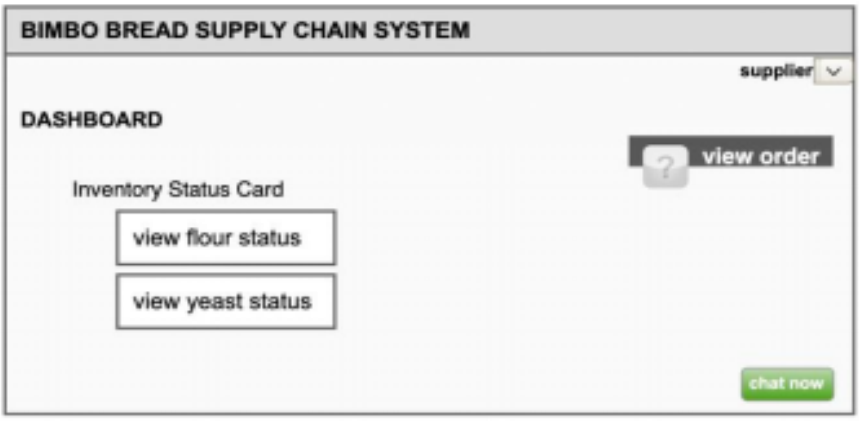

Figure 6. 2 Supply dashboard screen image

Figure 6.2 displays the supplier's operational dashboard, including real-time raw material stock levels (e.g., flour, yeast), pending order notifications, and an integrated chat panel for communication with bakeries

3. Bakery Production Dashboard

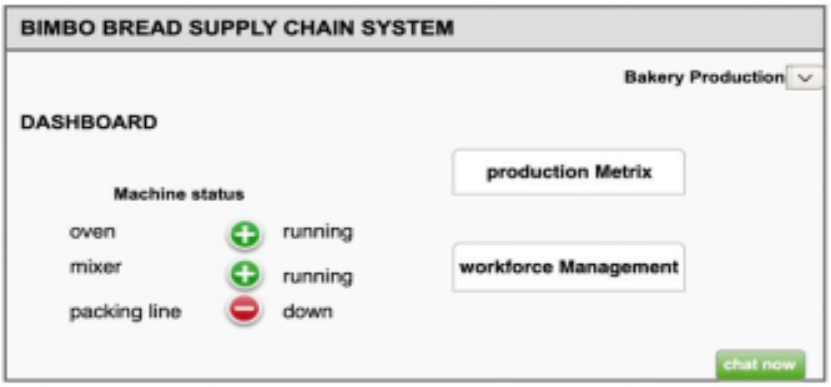

Figure 6. 3 Bakery production dashboard screen image

Figure 6.3 Illustrates the production management interface for bakery managers, highlights daily production targets versus actual output, machine status indicators, and workforce allocation charts.

4. Distribution Tracking Interface

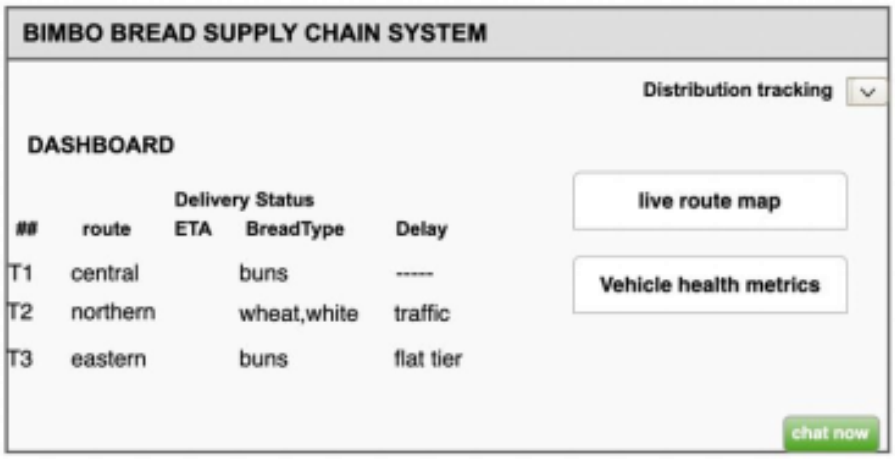

Figure 6. 4 Distribution Tracking interface screen image

Figure 6.4 depicts the distribution team's tracking screen, featuring a map view for route optimization, real-time vehicle locations, and delivery confirmation options

5. RETAIL MANAGER DASHBOARD

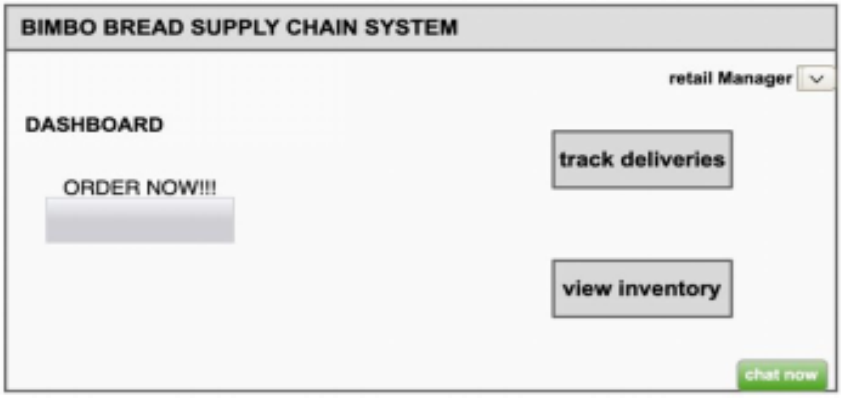

Figure 6. 5 Retail manager dashboard screen image

Figure 6.5 shows the retail manager's ordering interface, including a catalog of available bread products, inventory level indicators, and demand forecast displays to guide order placement

6.3 Screen Objects and Actions

Table 6. 1 Table of screen objects and their actions

figure	Screen Name	Object	Action	Result
6.1	Administrator Login	Username	Enter username	Input field accepts text
		Password field	Enter password	Masks input for security
		Login Button	Click	Authenticate or show error
6.2	Supplier Dashboard	Raw material stock panel	View	Displays inventory levels
		Pending orders alert	Click	Show order details
		Chat panel icon	Click	Opens communication panel
6.3	Bakery Production dashboard	Production gauge	View	Shows progress Vs targets
		Machine status indicator	Hover	Displays maintenance history
		Export report button	Click	Generates production report

6.4	Distribution Tracking	Live map view	Click	Shows delivery details
		Delivery confirm button	Click	Marks order as delivery
		Route optimize button	Click	Suggest delivery paths
6.5	Retail Ordering Portal	Bread catalog	Click	Show details (price, stock)
		Quantity selector	Adjust Unit	Updates order subtotal
		Checkout button	Click	Opens payment screen