Warehouse Management System

PennWest California

Design Document

CSC 4900 – Senior Project 1

Group Members:

Brian Colditz

Shakear Wilson

Ty Kress

William Serowik

Instructor Comments/Evaluation

Table of Contents

Abstract

The Warehouse Management System intends to help warehouse managers always track and manage their inventory; this includes ongoing shipments, deliveries, and supplies within the warehouse. While existing systems are available, they are less affordable and rely on manual or outdated systems to track stock, leading to inefficiencies, errors, and increased operational costs. The team aims to develop a warehouse management system to simplify inventory tracking and management. It will focus on user-friendliness, compatibility, and practical use by warehouse managers and workers. Implementing this system will allow these businesses to streamline operations, reduce errors, and increase productivity. This document will show the development management systems and the project plan.

Description of the Document

Purpose and Use:

The purpose of this document is to give the development team a strong foundation to reference when developing this project. This will help aid and guide them through the functionalities and goals of this warehouse management system as well as keeping everyone on track without deviating too much from the source. That will include detailed descriptions of all the software and functions needed for the project to work effectively and efficiently.

Ties to the Specification document:

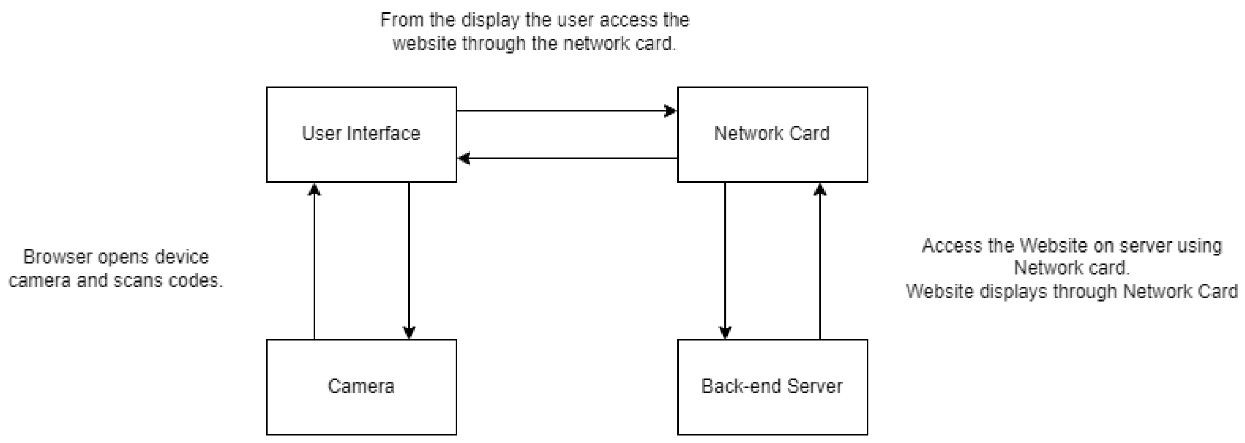
This document is basically an extension of the specification document as it will go more in-depth with this functionality and how we plan to execute that set plan. The document will show in detail how we will design a WMS that can properly and efficiently store, delete, and present the data to the user. It will be used by the developer team to implement its ideas into the project.

Intended Audience:

This document is primarily aimed at developers and designers based on its technical content and terminology. It isn't intended for clients but might benefit from reviewing to ensure alignment with client requirements. Developers and designers can use this document to gain a comprehensive understanding of the WMS which includes its front-end, back-end, data flow, functionality, modules, architecture, and system requirements.

Project Block Diagram

The WMS project will consist of two parts: a web browser that resides on a smartphone or computer (either Android, iOS, windows, or other), and backend code on a Linux or Windows server. The phone will interact with the backend server by sending scanned code info and view the website on a browser. Once the backend has received the code info, it will then process the data and update the database and webpage. From there the client can interact with the webpage to view the data stored for the WMS system.


From the display the user access the website through the network card.
User Interface
Network Card
Browser opens device camera and scans codes.
Access the Website on server using Network card.
Website displays through Network Card
Camera
Back-end Server

Design Details

<u>System Modules and responsibilities:</u>

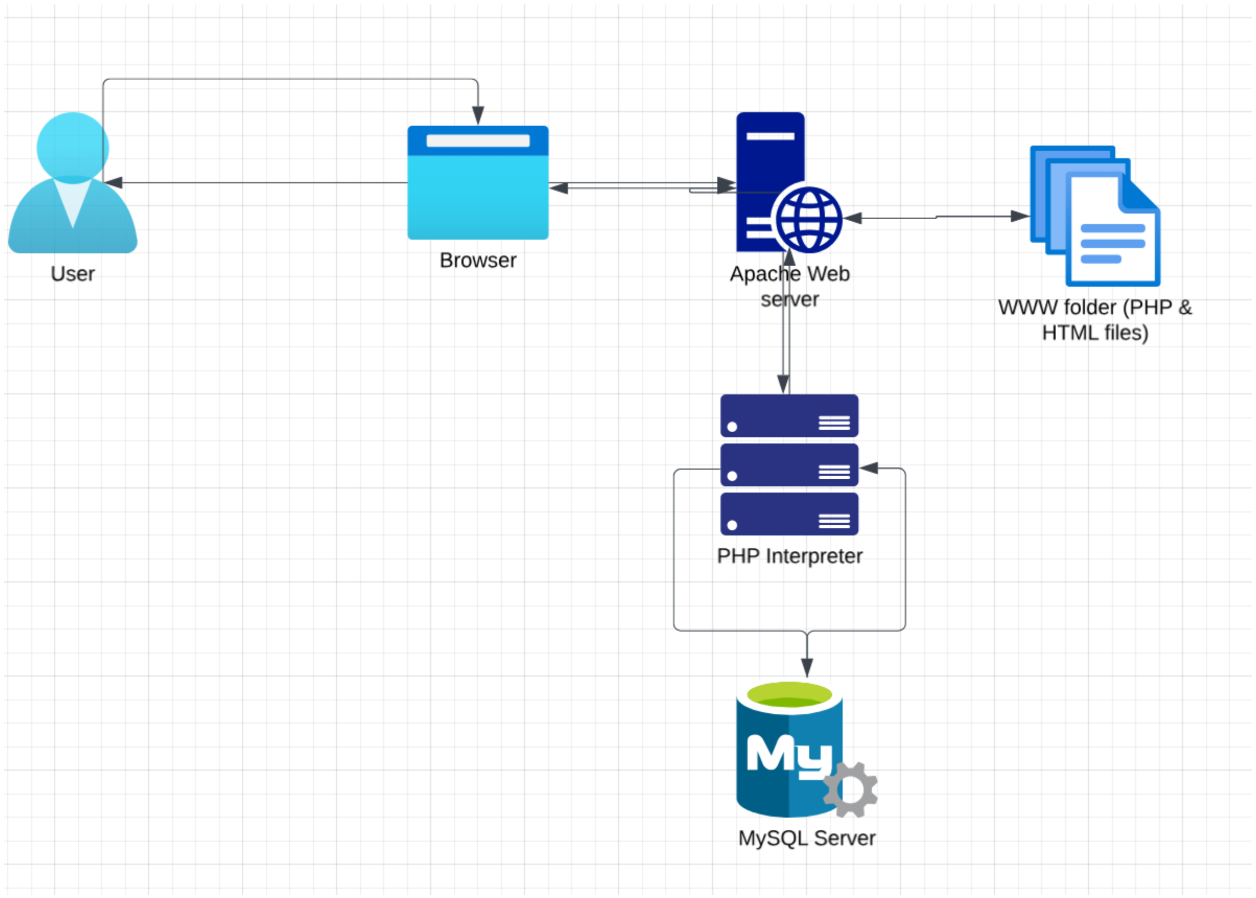
System Overview

The WMS is intended to streamline warehouse chores by offering a responsive, user-friendly interface that works on both desktop and mobile platforms. Its architecture separates user interface design, core application functionality, and backend data processing, making for easy maintenance and scalability. The system uses technologies such as Bootstrap for interactivity and PHP and MySQL for data management. Docker deployment ensures a consistent environment while also simplifying installation and upgrades.

Overview

The Warehouse Management System (WMS) will be built on a modular architecture to ensure flexibility and scalability. The system will be built using a responsive design to support both desktop and mobile platforms, making it accessible to a wide range of users. PHP and MySQL will provide the backend functionality, allowing for quick data administration and real-time changes via AJAX. Inventory procedures will be streamlined using barcode and QR scanning capabilities, and user access will be secured with role-based authentication. The system will be deployed in a Docker environment to ensure a constant and easily replicable setup throughout the development, testing, and production phases. This organized method assures that the WMS is reliable, user-friendly, and adaptable to future changes.

Architectural Diagram



Module Cohesion

Each module in the WMS is dedicated to a specific, well-defined task, resulting in excellent cohesiveness. For example, the Inventory Management Module is only dedicated to stock monitoring, whereas the Reporting Module is in charge of analytics and insights. This separation simplifies debugging, testing, and modification because changes to one module do not affect the functioning of others. The obvious separation of duties also makes it easier to improve or replace particular modules in the future.

Module Coupling

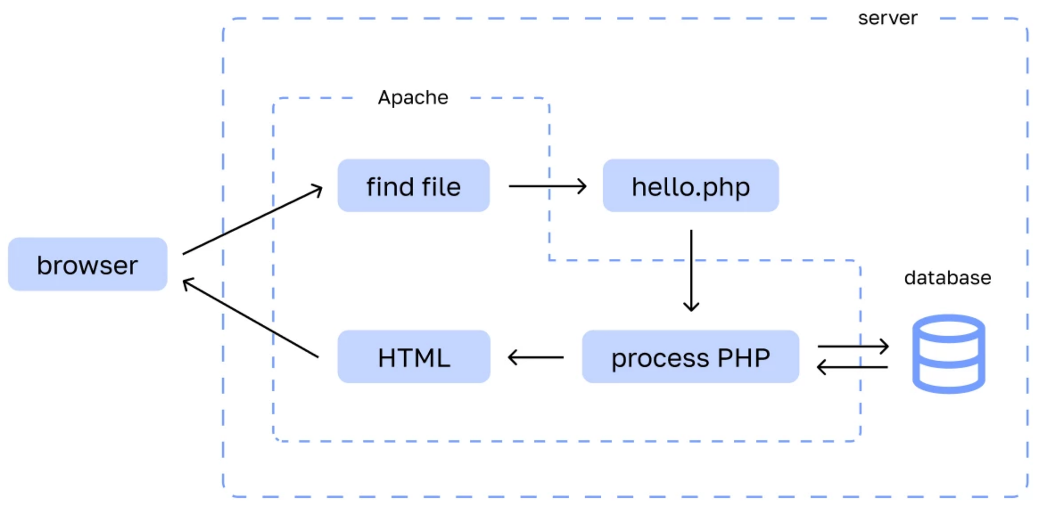
The WMS uses loose connectivity between modules to maximize flexibility and adaptability. Modules communicate with one other through established interfaces, allowing them to function independently. For example, the User Management Module secures access without requiring extensive connection with the Inventory or Order Modules. Similarly, the Reporting Module retrieves and processes data from other modules without relying directly on their internal designs. This method eliminates dependency and increases system sturdiness.

Design Analysis

Modularity, scalability, and maintainability are key design considerations for the WMS. By separating interface, functionality, and data processing levels, the system remains neat and responsive to changing requirements. Bootstrap, for example, allows for a responsive and appealing interface, whilst Docker ensures consistency in deployment. The backend sticks to object-oriented concepts, allowing the system to be extended or modified without disturbing its general structure.

Data Flow or Transaction Analysis

The flow of data within the WMS is intended to maximize efficiency while maintaining data integrity. When a user scans a QR code, the system retrieves product information from the database and changes inventory records in real time. Data from orders and shipments is processed to change stock levels, establish delivery schedules, and alert users of updates. Each transaction is validated to verify accuracy and security, resulting in an easy process for end users.



Design Organization

Detailed tabular description of Classes/Objects:

Class name: Categories

Class description: Organizes products into categories, with information about each category's name and description for better inventory management.

Class data members: int $id, string $name, string description

Class member functions: getId(), setId(int $id), getName(), setName(string $name), getDescription(), setDescription

Class name: Customers

Class description: Tracks customer information, including names, addresses, contact details, and associated user agents.

Class data members: int $id, string $name, string $address, string $email, string $phone, int $agent_user_id

Class member functions: getId(), setId(), getName(), setName(), getAddress(), setAddress(), getEmail(), setEmail(), getPhone(), setPhone(), getAgentUserId(), setAgentUserId()

Class name: Orders

Class description: Represents customer orders, capturing details like customer and agent IDs, order dates, statuses, and associated truck information.

Class data members: int $id, int $customer_id, int $user_agent_id, string $order_date, string $status, int $truck_id

Class member functions: getId(), setId(), getCustomerId(), setCustomerId(), getUserAgentId(), setUserAgentId(), getOrderDate(), setOrderDate(), getStatus(), setStatus(), getTruckId(), setTruckId()

Class name: OrderItems

Class description: Tracks items within an order, including product details, quantities, and their association with specific orders.

Class data members: int $id, int $order_id, int $product_id, int $quantity

Class member functions: getId(), setId(), getOrderId(), setOrderId(), getProductId(), setProductId(), getQuantity(), setQuantity()

Class name: InventoryAudits

Class description: Handles data related to inventory audits, including expected and actual quantities, discrepancies, audit dates, and location details.

Class data members: int $id, int $user_id, int $product_id, int $expected_quantity, int $actual_quantity, int $discrepancy, string $audit_date, int $location_detail_id

Class member functions: getId(), setId(), getUserId(), setUserId(), getProductId(), setProductId(), getExpectedQuantity(), setExpectedQuantity(), getActualQuantity(), setActualQuantity(int $actual_quantity), getDiscrepancy(), setDiscrepancy(), getAuditDate(), setAuditDate(), getLocationDetailId(), setLocationDetailId()

Class name: VendorOrders

Class description: Manages orders placed with vendors, tracking vendor details, order dates, statuses, and associated truck information.

Class data members: int $id, int $vendor_id, string $order_date, string $status, int $truck_id

Class member functions: getId(), setId(), getVendorId(), setVendorId(), getOrderDate(), setOrderDate(), getStatus(), setStatus(), getTruckId(), setTruckId()

Class name: VendorOrderItems

Class description: Tracks individual items in vendor orders, including the associated order, product details, and quantities.

Class data members: int $id, int $order_id, int $product_id, int $quantity

Class member functions: getId(), setId(), getOrderId(), setOrderId(), getProductId(), setProductId(), getQuantity(), setQuantity()

Class name: Racks

Class description: Represents storage racks in the warehouse, including attributes such as color and unique identification.

Class data members: int $id, string $color

Class member functions: getId(), setId(), getColor(), setColor()

Class name: Colors

Class description: Maintains a list of available color options used for categorizing or identifying warehouse items or storage units.

Class data members: int $id, string $name

Class member functions: getId(), setId(), getName(), setName()

Class name: Products

Class description: Contains information about products stored in the warehouse, including names, descriptions, prices, and categories.

Class data members: int $id, string $name, string $description, float $price, int $category_id

Class member functions: getId(), setId(), getName(), setName(), getDescription(), setDescription(), getPrice(), setPrice(), getCategoryId(), setCategoryId()

Class name: Vendors

Class description: Stores vendor details, including contact information such as names, emails, and phone numbers.

Class data members: int $id, string $name, string $email, string $phone

Class member functions: getId(), setId(), getName(), setName(), getEmail(), setEmail(), getPhone(), setPhone()

Class name: Trucks

Class description: Tracks vehicles used for transportation, including license plates and driver information.

Class data members: int $id, string $license_plate, string $driver_name

Class member functions: getId(), setId(), getLicensePlate(), setLicensePlate(), getDriverName(), setDriverName()

Class name: Users

Class description: Manages system users, including usernames, passwords, and roles within the warehouse management system.

Class data members: int $id, string $username, string $password, string $role

Class member functions: getId(), setId(), getUsername(), setUsername(), getPassword(), setPassword(), getRole(), setRole()

Class name: UserRoles

Class description: Defines various roles available in the system to control user permissions and access.

Class data members: int $id, string $name

Class member functions: getId(), setId(), getName(), setName()

Class name: InventoryTransactions

Class description: Records changes to inventory, detailing the type of change, quantities involved, and transaction dates.

Class data members: int $id, int $product_id, int $user_id, string $change_type, int $quantity, int $previous_quantity, int $new_quantity, string $transaction_date, int $location_detail_id

Class member functions: getId(), setId(), getProductId(), setProductId(), getUserId(), setUserId(), getChangeType(), setChangeType(), getQuantity(), setQuantity(), getPreviousQuantity(), setPreviousQuantity(), getNewQuantity(), setNewQuantity(), getTransactionDate(), setTransactionDate(), getLocationDetailId(), setLocationDetailId()

Class name: Locations

Class description: Represents specific storage locations in the warehouse, including rack IDs and bay details.

Class data members: int $id, int $rack_id, string $bay

Class member functions: getId(), setId(), getRackId(), setRackId(), getBay(), setBay()

Functional description

Categories Class

getId()

Input: None.

Output: Retrieves the ID of the category.

Return Parameters: Returns the category ID as an integer.

Types: Integer data type is used.

setId(int $id)

Input: Accepts an integer ID to set for the category.

Output: None.

Return Parameters: None.

Types: Integer data type is used

getName()

Input: None.

Output: Retrieves the name of the category.

Return Parameters: Returns the category name as a string.

Types: String data type is used.

setName(string $name)

Input: Accepts a string to set as the category name.

Output: None.

Return Parameters: None.

Types: String data type is used.

getDescription()

Input: None.

Output: Retrieves the description of the category.

Return Parameters: Returns the description as a string.

Types: String data type is used.

setDescription(string $description)

Input: Accepts a string to set as the category description.

Output: None.

Return Parameters: None.

Types: String data type is used.

Customers Class

getId()

Input: None.

Output: Retrieves the ID of the customer.

Return Parameters: Returns the customer ID as an integer.

Types: Integer data type is used.

setId(int $id)

Input: Accepts an integer ID to set for the customer.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getName()

Input: None.

Output: Retrieves the name of the customer.

Return Parameters: Returns the customer name as a string.

Types: String data type is used.

setName(string $name)

Input: Accepts a string to set as the customer name.

Output: None.

Return Parameters: None.

Types: String data type is used.

getAddress()

Input: None.

Output: Retrieves the address of the customer.

Return Parameters: Returns the address as a string.

Types: String data type is used.

setAddress(string $address)

Input: Accepts a string to set as the customer address.

Output: None.

Return Parameters: None.

Types: String data type is used.

getEmail()

Input: None.

Output: Retrieves the email address of the customer.

Return Parameters: Returns the email as a string.

Types: String data type is used.

setEmail(string $email)

Input: Accepts a string to set as the customer's email address.

Output: None.

Return Parameters: None.

Types: String data type is used.

getPhone()

Input: None.

Output: Retrieves the phone number of the customer.

Return Parameters: Returns the phone number as a string.

Types: String data type is used.

setPhone(string $phone)

Input: Accepts a string to set as the customer's phone number.

Output: None.

Return Parameters: None.

Types: String data type is used.

getAgentUserId()

Input: None.

Output: Retrieves the ID of the agent associated with the customer.

Return Parameters: Returns the agent user ID as an integer.

Types: Integer data type is used.

setAgentUserId(int $agent_user_id)

Input: Accepts an integer ID to set as the agent for the customer.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

Orders Class

getId()

Input: None.

Output: Retrieves the ID of the order.

Return Parameters: Returns the order ID as an integer.

Types: Integer data type is used.

setId(int $id)

Input: Accepts an integer ID to set for the order.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getCustomerId()

Input: None.

Output: Retrieves the customer ID associated with the order.

Return Parameters: Returns the customer ID as an integer.

Types: Integer data type is used.

setCustomerId(int $customer_id)

Input: Accepts an integer customer ID to associate with the order.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getUserAgentId()

Input: None.

Output: Retrieves the user agent ID associated with the order.

Return Parameters: Returns the user agent ID as an integer.

Types: Integer data type is used.

setUserAgentId(int $user_agent_id)

Input: Accepts an integer user agent ID to associate with the order.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getOrderDate()

Input: None.

Output: Retrieves the date of the order.

Return Parameters: Returns the order date as a string.

Types: String data type is used.

setOrderDate(string $order_date)

Input: Accepts a string to set as the order date.

Output: None.

Return Parameters: None.

Types: String data type is used.

getStatus()

Input: None.

Output: Retrieves the status of the order.

Return Parameters: Returns the order status as a string.

Types: String data type is used.

setStatus(string $status)

Input: Accepts a string to set as the order status.

Output: None.

Return Parameters: None.

Types: String data type is used.

getTruckId()

Input: None.

Output: Retrieves the truck ID associated with the order.

Return Parameters: Returns the truck ID as an integer.

Types: Integer data type is used.

setTruckId(int $truck_id)

Input: Accepts an integer truck ID to associate with the order.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

OrderItems Class

getId()

Input: None.

Output: Retrieves the ID of the order item.

Return Parameters: Returns the order item ID as an integer.

Types: Integer data type is used.

setId(int $id)

Input: Accepts an integer ID to set for the order item.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getOrderId()

Input: None.

Output: Retrieves the order ID associated with the order item.

Return Parameters: Returns the order ID as an integer.

Types: Integer data type is used.

setOrderId(int $order_id)

Input: Accepts an integer order ID to associate with the order item.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getProductId()

Input: None.

Output: Retrieves the product ID associated with the order item.

Return Parameters: Returns the product ID as an integer.

Types: Integer data type is used.

setProductId(int $product_id)

Input: Accepts an integer product ID to associate with the order item.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getQuantity()

Input: None.

Output: Retrieves the quantity of the product in the order item.

Return Parameters: Returns the quantity as an integer.

Types: Integer data type is used.

setQuantity(int $quantity)

Input: Accepts an integer to set as the quantity of the product.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

Inventory Audits Class

getId()

Input: None

Output: Returns the unique identifier of the inventory audit.

Return Parameters: An integer representing the audit ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the audit ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getUserId()

Input: None

Output: Returns the user ID associated with the audit.

Return Parameters: An integer representing the user ID.

Types: Integer data type is used.

setUserId(int $user_id)

Input: Takes an integer `user_id` to set the user ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getProductId()

Input: None

Output: Returns the product ID associated with the inventory audit.

Return Parameters: An integer representing the product ID.

Types: Integer data type is used.

setProductId(int $product_id)

Input: Takes an integer `product_id` to set the product ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getExpectedQuantity()

Input: None

Output: Returns the expected quantity of the product in the inventory.

Return Parameters: An integer representing the expected quantity.

Types: Integer data type is used.

setExpectedQuantity(int $expected_quantity)

Input: Takes an integer `expected_quantity` to set the expected quantity.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getActualQuantity()

Input: None

Output: Returns the actual quantity of the product in the inventory.

Return Parameters: An integer representing the actual quantity.

Types: Integer data type is used.

setActualQuantity(int $actual_quantity)

Input: Takes an integer `actual_quantity` to set the actual quantity.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getDiscrepancy()

Input: None

Output: Returns the discrepancy between expected and actual quantity.

Return Parameters: An integer representing the discrepancy.

Types: Integer data type is used.

setDiscrepancy(int $discrepancy)

Input: Takes an integer `discrepancy` to set the discrepancy value.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getAuditDate()

Input: None

Output: Returns the date when the audit took place.

Return Parameters: A string representing the audit date.

Types: String data type is used.

setAuditDate(string $audit_date)

Input: Takes a string `audit_date` to set the audit date.

Output: None.

Return Parameters: None.

Types: String data type is used.

getLocationDetailId()

Input: None

Output: Returns the location detail ID related to the audit.

Return Parameters: An integer representing the location detail ID.

Types: Integer data type is used.

setLocationDetailId(int $location_detail_id)

Input: Takes an integer `location_detail_id` to set the location detail ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

VendorOrders Class

getId()

Input: None

Output: Returns the unique identifier of the vendor order.

Return Parameters: An integer representing the order ID.

Types: Integer data type is used

setId(int $id)

Input: Takes an integer `id` to set the vendor order ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getVendorId()

Input: None

Output: Returns the vendor ID associated with the order.

Return Parameters: An integer representing the vendor ID.

Types: Integer data type is used.

setVendorId(int $vendor_id)

Input: Takes an integer `vendor_id` to set the vendor ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getOrderDate()

Input: None

Output: Returns the date when the vendor order was placed.

Return Parameters: A string representing the order date.

Types: String data type is used.

setOrderDate(string $order_date)

Input: Takes a string `order_date` to set the order date.

Output: None.

Return Parameters: None.

Types: String data type is used.

getStatus()

Input: None

Output: Returns the status of the vendor order.

Return Parameters: A string representing the order status.

Types: String data type is used..

setStatus(string $status)

Input: Takes a string `status` to set the order status.

Output: None.

Return Parameters: None.

Types: String data type is used.

getTruckId()

Input: None

Output: Returns the truck ID associated with the vendor order.

Return Parameters: An integer representing the truck ID.

Types: Integer data type is used.

setTruckId(int $truck_id)

Input: Takes an integer `truck_id` to set the truck ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

VendorOrderItems Class

getId()

Input: None

Output: Returns the unique identifier of the vendor order item.

Return Parameters: An integer representing the order item ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the vendor order item ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getOrderId()

Input: None

Output: Returns the vendor order ID associated with the order item.

Return Parameters: An integer representing the order ID.

Types: Integer data type is used.

setOrderId(int $order_id)

Input: Takes an integer `order_id` to set the order ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getProductId()

Input: None

Output: Returns the product ID associated with the order item.

Return Parameters: An integer representing the product ID.

Types: Integer data type is used.

setProductId(int $product_id)

Input: Takes an integer `product_id` to set the product ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getQuantity()

Input: None

Output: Returns the quantity of the product in the order item.

Return Parameters: An integer representing the quantity.

Types: Integer data type is used.

setQuantity(int $quantity)

Input: Takes an integer `quantity` to set the product quantity.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

Racks Class

getId()

Input: None

Output: Returns the unique identifier of the rack.

Return Parameters: An integer representing the rack ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the rack ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getColor()

Input: None

Output: Returns the color of the rack.

Return Parameters: A string representing the color.

Types: String data type is used.

setColor(string $color)

Input: Takes a string `color` to set the rack color.

Output: None.

Return Parameters: None.

Types: String data type is used.

Colors Class

getId()

Input: None

Output: Returns the unique identifier of the color.

Return Parameters: An integer representing the color ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the color ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getName()

Input: None

Output: Returns the name of the color.

Return Parameters: A string representing the color name.

Types: String data type is used.

setName(string $name)

Input: Takes a string `name` to set the color name.

Output: None.

Return Parameters: None.

Types: String data type is used.

Products Class

getId()

Input: None

Output: Returns the unique identifier of the product.

Return Parameters: An integer representing the product ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the product ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getName()

Input: None

Output: Returns the name of the product.

Return Parameters: A string representing the product name.

Types: String data type is used.

setName(string $name)

Input: Takes a string `name` to set the product name.

Output: None.

Return Parameters: None.

Types: String data type is used.

getDescription()

Input: None

Output: Returns the description of the product.

Return Parameters: A string representing the product description.

Types: String data type is used.

setDescription(string $description)

Input: Takes a string `description` to set the product description.

Output: None.

Return Parameters: None.

Types: String data type is used.

getPrice()

Input: None

Output: Returns the price of the product.

Return Parameters: A floating-point number representing the product price.

Types: Float data type is used.

setPrice(float $price)

Input: Takes a float `price` to set the product price.

Output: None.

Return Parameters: None.

Types: Float data type is used.

getCategoryId()

Input: None

Output: Returns the category ID associated with the product.

Return Parameters: An integer representing the category ID.

Types: Integer data type is used.

setCategoryId(int $category_id)

Input: Takes an integer `category_id` to set the product category ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

Vendors Class

getId()

Input: None

Output: Returns the unique identifier of the vendor.

Return Parameters: An integer representing the vendor ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the vendor ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getName()

Input: None

Output: Returns the name of the vendor.

Return Parameters: A string representing the vendor name.

Types: String data type is used.

setName(string $name)

Input: Takes a string `name` to set the vendor name.

Output: None.

Return Parameters: None.

Types: String data type is used.

getEmail()

Input: None

Output: Returns the email address of the vendor.

Return Parameters: A string representing the vendor email.

Types: String data type is used.

setEmail(string $email)

Input: Takes a string `email` to set the vendor email.

Output: None.

Return Parameters: None.

Types: String data type is used.

getPhone()

Input: None

Output: Returns the phone number of the vendor.

Return Parameters: A string representing the vendor phone number.

Types: String data type is used.

setPhone(string $phone)

Input: Takes a string `phone` to set the vendor phone number.

Output: None.

Return Parameters: None.

Types: String data type is used.

Trucks Class

getId()

Input: None

Output: Returns the unique identifier of the truck.

Return Parameters: An integer representing the truck ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the truck ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getLicensePlate()

Input: None

Output: Returns the license plate number of the truck.

Return Parameters: A string representing the truck’s license plate.

Types: String data type is used.

setLicensePlate(string $license_plate)

Input: Takes a string `license_plate` to set the truck's license plate number.

Output: None.

Return Parameters: None.

Types: String data type is used.

getDriverName()

Input: None

Output: Returns the name of the truck driver.

Return Parameters: A string representing the driver’s name.

Types: String data type is used.

setDriverName(string $driver_name)

Input: Takes a string `driver_name` to set the driver’s name.

Output: None.

Return Parameters: None.

Types: String data type is used.

Users Class

getId()

Input: None

Output: Returns the unique identifier of the user.

Return Parameters: An integer representing the user ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the user ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getUsername()

Input: None

Output: Returns the username of the user.

Return Parameters: A string representing the username.

Types: String data type is used.

setUsername(string $username)

Input: Takes a string `username` to set the username.

Output: None.

Return Parameters: None.

Types: String data type is used.

getPassword()

Input: None

Output: Returns the password of the user.

Return Parameters: A string representing the password.

Types: String data type is used.

setPassword(string $password)

Input: Takes a string `password` to set the password.

Output: None.

Return Parameters: None.

Types: String data type is used.

getRole()

Input: None

Output: Returns the role of the user.

Return Parameters: A string representing the user's role.

Types: String data type is used.

setRole(string $role)

Input: Takes a string `role` to set the user’s role.

Output: None.

Return Parameters: None.

Types: String data type is used.

UserRoles Class

getId()

Input: None

Output: Returns the unique identifier of the user role.

Return Parameters: An integer representing the user role ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the user role ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getName()

Input: None

Output: Returns the name of the user role.

Return Parameters: A string representing the role name.

Types: String data type is used.

setName(string $name)

Input: Takes a string `name` to set the role name.

Output: None.

Return Parameters: None.

Types: String data type is used.

InventoryTransactions Class

getId()

Input: None

Output: Returns the unique identifier of the inventory transaction.

Return Parameters: An integer representing the transaction ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the transaction ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getProductId()

Input: None

Output: Returns the product ID associated with the inventory transaction.

Return Parameters: An integer representing the product ID.

Types: Integer data type is used.

setProductId(int $product_id)

Input: Takes an integer `product_id` to set the product ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getUserId()

Input: None

Output: Returns the user ID associated with the transaction.

Return Parameters: An integer representing the user ID.

Types: Integer data type is used.

setUserId(int $user_id)

Input: Takes an integer `user_id` to set the user ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getChangeType()

Input: None

Output: Returns the type of change made in the transaction (e.g., "addition", "removal").

Return Parameters: A string representing the change type.

Types: String data type is used.

setChangeType(string $change_type)

Input: Takes a string `change_type` to set the type of change.

Output: None.

Return Parameters: None.

Types: String data type is used.

getQuantity()

Input: None

Output: Returns the quantity involved in the transaction.

Return Parameters: An integer representing the quantity.

Types: Integer data type is used.

setQuantity(int $quantity)

Input: Takes an integer `quantity` to set the transaction quantity.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getPreviousQuantity()

Input: None

Output: Returns the previous quantity before the transaction.

Return Parameters: An integer representing the previous quantity.

Types: Integer data type is used.

setPreviousQuantity(int $previous_quantity)

Input: Takes an integer `previous_quantity` to set the previous quantity.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getNewQuantity()

Input: None

Output: Returns the new quantity after the transaction.

Return Parameters: An integer representing the new quantity.

Types: Integer data type is used.

setNewQuantity(int $new_quantity)

Input: Takes an integer `new_quantity` to set the new quantity.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getTransactionDate()

Input: None

Output: Returns the date of the transaction.

Return Parameters: A string representing the transaction date.

Types: String data type is used.

setTransactionDate(string $transaction_date)

Input: Takes a string `transaction_date` to set the transaction date.

Output: None.

Return Parameters: None.

Types: String data type is used.

getLocationDetailId()

Input: None

Output: Returns the location detail ID associated with the transaction.

Return Parameters: An integer representing the location detail ID.

Types: Integer data type is used.

setLocationDetailId(int $location_detail_id)

Input: Takes an integer `location_detail_id` to set the location detail ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

Locations Class

getId()

Input: None

Output: Returns the unique identifier of the location.

Return Parameters: An integer representing the location ID.

Types: Integer data type is used.

setId(int $id)

Input: Takes an integer `id` to set the location ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getRackId()

Input: None

Output: Returns the rack ID associated with the location.

Return Parameters: An integer representing the rack ID.

Types: Integer data type is used.

setRackId(int $rack_id)

Input: Takes an integer `rack_id` to set the rack ID.

Output: None.

Return Parameters: None.

Types: Integer data type is used.

getBay()

Input: None

Output: Returns the bay location within the rack.

Return Parameters: A string representing the bay.

Types: String data type is used.

setBay(string $bay)

Input: Takes a string bay to set the bay location.

Output: None.

Return Parameters: None.

Types: String data type is used.

Files accessed

As of now, no data will be stored on any handheld devices. All data will be on a server. Including the database and application functions.

Real-time requirements

The program needs to run in a reasonably speedy amount of time. If the application is too slow to retrieve, add, or edit data, there could be delays or users could get frustrated and look for alternatives. If the application is locally hosted connection to the LAN is necessary and if hosted remotely an internet connection is required. Ideally when remote, it would be as fast as if the interaction was on a local system.

Messages

001	Invalid Data
002	Incomplete Data
003	Product Does Not Exist
004	Item Does Not Exist
005	Location Does Not Exist
006	User Does Not Exist
007	Customer Does Not Exist
404	Page No longer Available

Narrative / PDL

Menu:

1. Product
2. Item
3. Location
4. Customer
5. Inventory

Product:

1. Add

    Input required data then returned to menu with confirmation.

2. Remove

    Input product to be removed then returned to menu with confirmation.

3. View

    View all products in table format.

Item:

1. Add

Input required data then returned to menu with confirmation.

2. Remove

   Input item to be removed then returned to menu with confirmation.

3. View

   View all items in table format.

Location:

1. Add

   Input required data then returned to menu with confirmation.

2. Remove

   Input required location to be removed then returned to menu with confirmation.

3. View

   View all locations in table format.

Customer:

1. Add

   Input required data then returned to menu with confirmation.

2. Remove

   Input required customer to be removed then returned to menu with confirmation.

3. View

   View all customers in table format.

Inventory:

1. View

   View all inventorys in table format.

Decision: Programming language/Reuse/Portability

The team decided on using PHP as the main programing language for this product. We chose it for a few reasons. There is a large number of public libraries that will improve development time and ease of integration. PHP also has the ability to generate and run html code in the same file allowing for all the code for one page to be in a single file. Being able to integrate html code allows us to use a website template for a quick and easy professional looking interface. One of our members has experience writing in PHP, Brain Colditz, he has worked on a number of other projects using the language and is able to help familiarize us with it. The PHP language is similar to other languages the rest of us have used so it will be a simple transition.

Implementation Timeline

December	January	February	March	April	May
Developing	Developing	Developing	Developing/ Testing	Developing/ Testing	Testing/ Present

Design Testing

Design testing will be done throughout the entire development process. This will be Done as each section is completed. Design testing will include peer code reviews. Each of us will review the written code to give constructive criticism as we are all computer science majors. Each member of the developing team will try to “break” our application. We will try to break the application by looking for any bugs that might exist and inputting incorrect data. This will ensure that our code is implemented correctly and that our application is operating as intended. The testing team will be the development team. This will enable quick real-time results.

Appendix A: Schematic & Bill of Materials (BOM)

We have no hardware costs.

The Website Template cost $20.

Appendix B: Team Details

The leader of this documentation was William Serowik. All members took part in proofreading, formatting, and grammar evaluations. Each member of the team completed their section as well

as helped to add to other members sections. The requirement document was created by all the following individual efforts:

Brian was responsible for Design Details, System Modules and responsibilities, System Overview, Architectural Diagram, Module Cohesion, Module Coupling, Design Analysis, and Data Flow or Transaction Analysis.

Shakear was responsible for Design Organization, Modules used, Files accessed, Real-time requirements, Messages, Narrative / PDL, and Decision: Programming language/Reuse/Portability.

Ty was responsible for Detailed tabular description of Classes/Objects, Description, Data members / types / constraints, Member function listing / description, and Functional description Input/output/return parameters / type.

William was responsible for Abstract, Description of the Document, Purpose and Use, Ties to the Specification document, Intended Audience, Project Block Diagram with description Implementation Timeline, and Design Testing.

Other contributions that were coordinated by group members consisted of:

- Discord meetings/discussion
- In person discussion

- Formatting and planning outline of document
- Meeting at Writing Center
- Feedback and Proofreading

Appendix C: Writing Center Report

The following notes related to your 12/10/2024 1:30 PM EST appointment with Caroline Krofcheck have been shared with you:

Student came into the writing center to review senior project for software engineering. We talked about formatting for the document including improving margins and spacing issues.

-Caroline Krofcheck
Foundry Writing Center Consultant

Appendix D: Workflow Authentication

I, Brian Colditz, agree with the details defined in this document that represent functional requirements of WMS. Also, I agree that the work that was done as stated by this document

Signature: Brian Colditz Date: 12/10/2024

I, Ty Kress, agree with the details defined in this document that represent functional requirements of WMS. Also, I agree that the work that was done as stated by this document

Signature: [illegible] Date: 12/10/2024

I, Shakear Wilson, agree with the details defined in this document that represent functional requirements of WMS. Also, I agree that the work that was done as stated by this document

Signature: [illegible] Date: 12/10/2024

I, William Serowik, agree with the details defined in this document that represent functional requirements of WMS. Also, I agree that the work that was done as stated by this document

Signature: [illegible] Date: 12/10/2024

Appendix E: References

Apache and Nginx Web Servers – Ispmanager blog. – Ispmanager Blog. (2023, March 14). https://www.ispmanager.com/news/apache-nginx-ols

Modular Architecture Software Development. TRIARE. (2024, October 27). https://triare.net/insights/modular-architecture-software/