

CPSC 471
KMS Tools Customer Assistant Database System

Final Report

Group 48

Vaibhav Kapoor, Marco Arias Barreto, Nolan Chan

Table of Contents

Abstract	2
Introduction	2
Project Design Section	3
ER Diagram	6
Project Implementation Section	7
SQL Statements for all Implemented Transactions	9
API Documentation	17
User Guide For KMS Tools Website	18
User Guide To Run Website/ API for testing	22
References	24

Abstract

E-commerce has had its share of the retail market grow steadily over the past couple of years. For example, in Q4 of 2019, Canadian e-commerce sales were up 17.3% year-over-year^[1], while location-based retail has only experienced a sales increase of 1.4% in the same period. This growth can be attributed to the fact that shopping online has a variety of advantages over shopping in a physical store that stems from a more convenient shopping experience for the customer. For example, a customer shopping online does not have to wander around in a physical store, to look for a specific product that they are looking for, or a sales associate to help them find the product. Despite these advantages, brick-and-mortar stores still have their place in the retail market as even though many Canadian brick-and-mortar based retailers have options where a customer can buy their products online, it is estimated that only 1.3%^[1] of their sales are attributed to online purchases. This highlights the fact that shopping at a physical location has its advantages such as a customer being able to see and hold a product before purchasing, and a customer being able to ask for expert advice from a sales associate. As such, it can be advantageous to a brick-and-mortar based retailer to try and adopt some of the unique advantages that shopping online has by having their own system that a customer can use to augment their shopping experience, while also retaining its natural advantages of being a physical location-based retailer. The goal of this project is to implement such a system.

Introduction

KMS Tools is a Canadian retailer of tools that prides itself on its excellent customer support that its sales associates provide to their customers. However, not all questions that a customer may have for a sales associate require a sales associate's expert knowledge of tools but can be solved by a database system. Such a database system was developed for this project, and it is designed to contain features to make the customer's shopping experience at a brick-and-mortar KMS Tools branch more convenient. Such features include the ability to look up basic information on a tool and its location in a store branch, the ability to view a specific sales associate's shifts, and a feature where a customer can request a transfer or hold on a specific product at a store branch.

Our database system consists of a SQL database, an API used to interact with the database and a website (not fully implemented) that a customer/admin would use to access the features of this database system (e.x. view a product's location in a store branch). For our API we developed a REST API using PHP for our SQL database. For each table in our database, our API implemented READ, READ_ONE, CREATE, UPDATE and DELETE requests which responded with JSON responses for data and messages to the user along with an HTTP response code to communicate the result of the requests. In our website, we took advantage of the HTTP response codes to identify the success/ failure of the operations performed by the user with the database.

Project Design Section

Our database system has two types of users: admin users (sales associates), and customers. (Note: Not all of the features mentioned below are implemented completely into the website)

An admin user, or sales associate working for KMS, has the ability to: login into the system with their username and password, add new tools to the system, update information about an existing tool, and browse and search tools in the system.

- For an admin's ability to login to the system, the admin would first navigate to the admin login page from the customer login page. Then the admin would enter their username and password in the appropriate fields, and then they would press the "Login" button. The system would then use the API to check if the user login credentials entered are valid. If the user has provided incorrect admin login information, an appropriate message would appear on the screen telling them so. Afterward, the user can try to login as an admin again. After the admin has successfully logged in, they would be automatically directed to the admin main page.
- For an admin's ability to add a new tool to the system, the admin would first navigate to the add item to the inventory page from the admin main page. From there, the admin can then enter in the new product's name, SKU, UPC, type, quantity in stock, and regular price. Afterward, the admin would press the "Add Item" button. The system would then use the API to try to insert a new product_type into the database with the entered information. If the item could not be inserted into the system (e.x. A tool already exists with the same SKU), then an appropriate message would appear and the admin can try entering different product information to try again. Otherwise, if the tool was successfully inserted into the system an appropriate message would appear on the screen saying so.
- For an admin's ability to update information about an existing tool in the system, the admin would navigate to the update item page from the admin main page. From there, the admin would enter the SKU of the product that they wish to modify, and then they would press the "Enter" button. The system would then use the API to retrieve information corresponding to the product_type that has the same SKU that was entered. If a tool with the SKU that the user has entered does not exist in the system, an appropriate message would appear on the screen and then the admin can try to enter a different SKU. Otherwise, the website would then display information about the tool with the SKU entered, and then the user can modify this information. After the user is satisfied with their changes, they would press the "Update" button. The system would then use the API to attempt updating the product information. If the product's information was not able to be successfully changed (e.x. user tried to change the SKU to another SKU that already exists in the system), an appropriate error message would appear on the screen and the user can then try changing their modifications to then try to update the product information again. Otherwise, if the product information was successfully updated in the system, an appropriate message would appear on the screen saying so.
- For an admin's ability to browse or search for a tool in the system, the admin would navigate to the browse items page from the admin main page. The system would then

use the API to retrieve all products in the system. The system would then display all of the tools in the system as a list. The admin can then enter in a product's SKU at the top of the page and then press the "Go" button to lookup more detailed information about a product. The system would then use the API to retrieve information corresponding to the product_type that has the same SKU as the one that was entered. If a tool with the SKU that the user has entered does not exist in the system, an appropriate error message would appear on the screen and then the admin can try again by entering a different SKU. Otherwise, more detailed information about the product would appear on the page. To search for the product's location in a store branch, the user can then select a store branch from a drop-down menu, and then the system would use the API to get the location of the product at the specified store branch. Afterward, a map of the store branch would pop up with the product's location highlighted. The admin can then search for different tools by entering another SKU at the field at the top of the page and then pressing "Go".

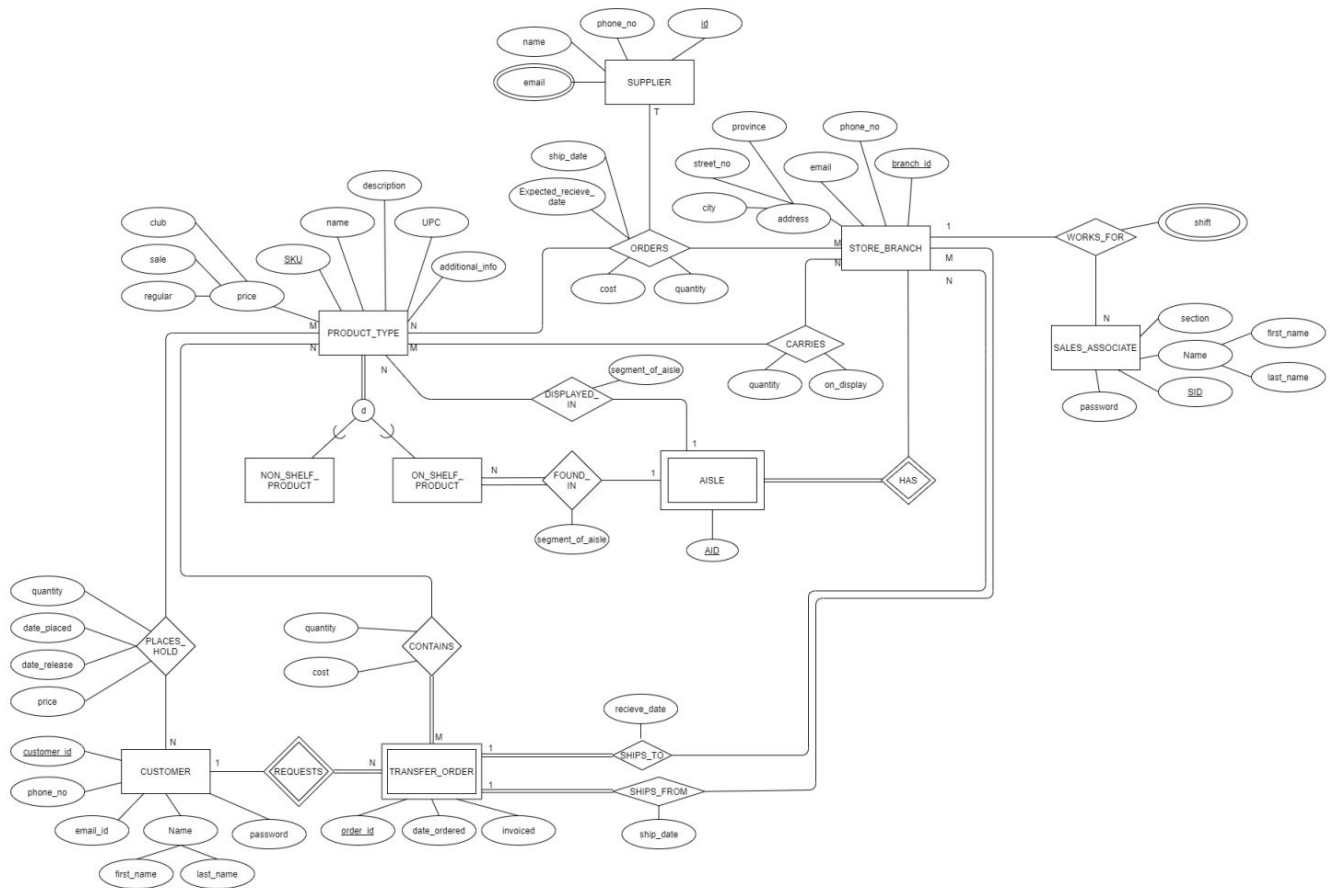
A customer has the ability to: login into the system with their username and password, register as a user by providing information such as their name and email, browse and search tools in the system, request a transfer or hold on a product at a specific store branch, and view a specific sales associate shift.

- For a customer's ability to login to the system, the customer would first open the webpage where they would be greeted by the customer login page. The customer would then enter their username and password in the appropriate fields. Afterward, the customer would press the "Login" button. The system would then use the API to check if the user login credentials entered are valid. If the customer has provided incorrect customer login details, a message would appear on the screen telling the customer so, and the customer can try to login again. After the customer has successfully logged in, they would be directed to the customer main page.
- For a customer's ability to register as a user, the customer would first navigate to the registration page from the customer login page. Then the customer can enter their username, first and last name, email, phone number, password, and their password again for confirmation. After the customer is satisfied with the information they typed in, they can then press the "Submit" button. The system would then use the API to attempt inserting a new customer into the database. If the customer could not be successfully registered as a user (i.e. username and password combination already exists in the system), an appropriate error message is displayed and the customer can try to register again. After the customer successfully registers as a user, they are automatically logged in and are redirected to the customer main page.
- For a customer's ability to browse or search for a tool in the system, the customer would navigate to the browse items page from the customer main page. The system would then use the API to retrieve all products in the system. The page would then display all of the tools in the system as a list. The customer can then enter a product's SKU at the top of the page and then press the "Go" button to look-up more detailed information about a product. The system would then use the API to retrieve information

corresponding to the product_type that has the same SKU as the one that was entered. If a tool with the SKU that the user has entered does not exist in the system, an appropriate error message would appear on the screen and then the customer can try again by entering a different SKU. Otherwise, more detailed information about the product would appear on the page. To search for the product's location in a store branch, the user would then select a store branch from a drop-down menu, and then the system would use the API to get the location of the product at the specified store branch. Afterward, a map of the store would pop up with the product's location highlighted. The customer can then search for different tools by entering another SKU at the field at the top of the page and then pressing "Go".

- For a customer's ability to request a transfer or hold on a product at a specific store branch, the customer would first navigate to the request hold/transfer page. The page would then have fields where a customer can enter the SKU of the product that they wish to place a hold/transfer on, how much of the product that they want to put on a hold/transfer(quantity), and the store branch that will hold the product for them. The system would then use the API to retrieve information about the product_type that has a SKU that matches the one the customer entered. If the customer entered a SKU that does not correspond to a product in the database, an appropriate error message would be displayed on the screen and the customer can try again by changing the SKU that they entered. If their request is successful, the system would then use the API to check if the specified store branch has enough of the product to fulfill the request. If the store branch does have enough of the product, the specified product would be put on a 'hold' for the customer. The system would then update the available quantity of the product via the API in the database and then a sales associate can fulfill the customer hold request. If the store branch does not have enough of the product to fulfill the request, the specified product would be 'transferred' (API would update the product quantity at the branch that the product is transferring from) from the nearest store branch that has enough of the product, where a sales associate would later complete this request. Regardless if the product is being put on hold or being transferred, a message is displayed on the screen containing important information such as: how long the products are on hold for (if it is a hold), or the estimated arrival date of the customer's transfer to their specified store branch.
- For a customer's ability to view a specific sales associate shifts, the customer would first navigate to the view sales associates shifts page from the customer main page. From there, the customer would select what KMS Tools branch they are interested in seeing the shifts of sales associates working there from a drop-down menu. The system would then use the API to retrieve the information of all sales associates working at the branch selected. Afterward, the page would then display all of the sales associates working at the selected store branch, what section that they work in (e.x. woodworking), and their weekly shifts.

ER Diagram



Assumptions:

1. Transfer orders must contain a product
2. Multiple customers can place holds of multiple products
3. A sales associate can only work at one store branch
4. Customers and sales associates are the only end-users and have login information
5. Each product type can only be found in one aisle in the store
6. Each product type can either be on store shelves (i.e. customers can take the item off of the store shelves by themselves and bring it to the cashier) or, the product type is not stored on store shelves (i.e. the product is large and is stored in the warehouse and a customer would have to ask an employee to help them purchase it)
7. Each product type has to have a unique SKU for the store
8. Any type of product can be displayed on a store aisle (even if it's a non-shelf product)

Changes to ERD:

- Suppliers were changed to only have one email address in order to simplify things (in diagram, email is a multi-valued attribute)
- Customers and Sales Associates were changed to have a username attribute for login purposes (before, it was intended that a customer or sales associate would use their unique id number as a username)

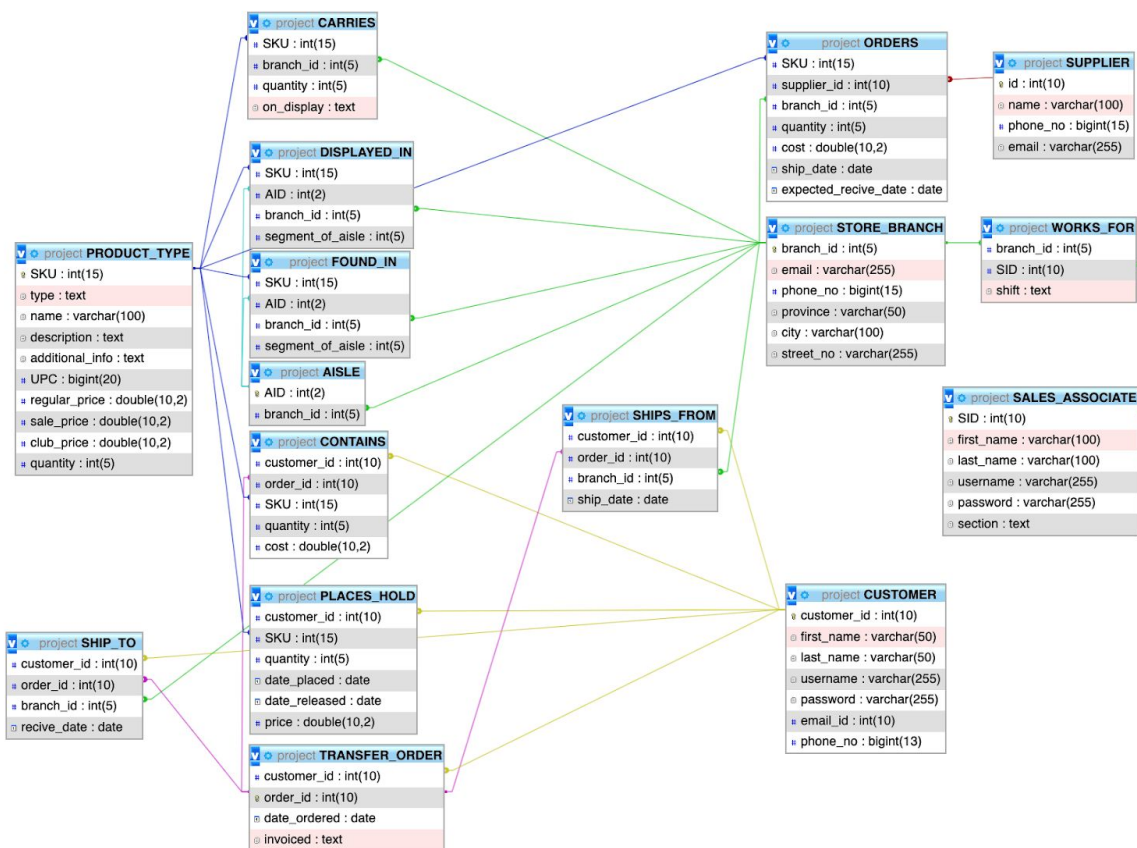
Relational Model Diagram:



(Note: This relational model diagram is different than the diagram submitted initially for this project on D2L as it includes the changes mentioned after the ER diagram)

phpMyAdmin on Apache

For the DBMS we used Apache to contain our databases as it contained both mysql and php. As it can be observed on the image below the sql database is based on the ER diagram shown before, this database can be found inside the file *project.sql*. Inside the file you can observe a total of 16 different databases which have different values as well as columns of data. For some of these turples we have placed foreign keys and primary keys for some values to be shared between databases.



SQL Statements for all Implemented Transactions

Aisle

Here `$table_name = "aisle";`

Read Aisle:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Aisle:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE AID=? LIMIT 0,1";
```

Create Aisle:

```
$query = "INSERT INTO " . $this->table_name. " SET AID=:AID, branch_id=:branch_id";
```

Update Aisle:

```
$query = "UPDATE " . $this->table_name. " SET branch_id=:branch_id WHERE  
AID=:AID";
```

Delete Aisle:

```
$query = "DELETE FROM " . $this->table_name. " WHERE AID=?";
```

Carries

Here `$table_name = "carries";`

Read Carries:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Carries:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE SKU=? LIMIT 0,1";
```

Create Carries:

```
$query = "INSERT INTO " . $this->table_name. " SET SKU=:SKU, branch_id=:branch_id,  
quantity=:quantity, on_display=:on_display";
```

Update Carries:

```
$query = "UPDATE " . $this->table_name. " SET branch_id=:branch_id,  
quantity=:quantity, on_display=:on_display WHERE SKU=:SKU";
```

Delete Carries:

```
$query = "DELETE FROM " . $this->table_name. " WHERE SKU=?";
```

Contains

Here `$table_name = "contains";`

Read Contains:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Contains:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE order_id=? LIMIT 0,1";
```

Create Contains:

```
$query = "INSERT INTO " . $this->table_name. " SET customer_id=:customer_id,  
order_id=:order_id, SKU=:SKU, quantity=:quantity, cost=:cost";
```

Update Contains:

```
$query = "UPDATE " . $this->table_name. " SET order_id = :order_id, SKU = :SKU,  
quantity = :quantity, cost = :cost WHERE customer_id = :customer_id";
```

Delete Contains:

```
$query = "DELETE FROM " . $this->table_name. " WHERE order_id=?";
```

Customer

Here `$table_name` = "customer";

Read Customer:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Customer:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE customer_id=? LIMIT 0,1";
```

Customer Authentication:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE username=@username  
AND password=@password";
```

Create Customer:

```
$query = "INSERT INTO " . $this->table_name. " SET customer_id=:customer_id,  
first_name=:first_name, last_name=:last_name, username=:username,  
password=:password, email_id=:email_id, phone_no=:phone_no";
```

Update Customer:

```
$query = "UPDATE " . $this->table_name. " SET first_name=:first_name,  
last_name=:last_name, username=:username, password=:password,  
email_id=:email_id, phone_no=:phone_no WHERE  
customer_id=:customer_id";
```

Delete Customer:

```
$query = "DELETE FROM " . $this->table_name. " WHERE customer_id=?";
```

Displayed_in

Here `$table_name = "displayed_in";`

Read Displayed_in:

`$query = "SELECT * FROM " . $this->table_name;`

Read One Displayed_in:

`$query = "SELECT * FROM " . $this->table_name. " WHERE SKU=? LIMIT 0,1";`

Create Displayed_in:

`$query = "INSERT INTO " . $this->table_name. " SET SKU=:SKU, AID=:AID,
branch_id=:branch_id, segment_of_aisle=:segment_of_aisle";`

Update Displayed_in:

`$query = "UPDATE" . $this->table_name. " SET SKU=:SKU, AID=:AID,
branch_id=:branch_id, segment_of_aisle=:segment_of_aisle";`

Delete Displayed_in:

`$query = "DELETE FROM " . $this->table_name. " WHERE SKU=?";`

Found_in

Here `$table_name = "found_in";`

Read Found_in:

`$query = "SELECT * FROM " . $this->table_name;`

Read One Found_in:

`$query = "SELECT * FROM " . $this->table_name. " WHERE SKU=? LIMIT 0,1";`

Create Found_in:

`$query = "INSERT INTO " . $this->table_name. " SET SKU=:SKU, AID=:AID,
branch_id=:branch_id, segment_of_aisle=:segment_of_aisle";`

Update Found_in:

`$query = "UPDATE" . $this->table_name. " SET SKU=:SKU, AID=:AID,
branch_id=:branch_id, segment_of_aisle=:segment_of_aisle";`

Delete Found_in:

`$query = "DELETE FROM " . $this->table_name. " WHERE SKU=?";`

Order

Here `$table_name = "order";`

Read Order:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Order:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE SKU=? LIMIT 0,1";
```

Create Order:

```
$query = "INSERT INTO " . $this->table_name. " SET SKU=:SKU,  
supplier_id=:supplier_id, branch_id=:branch_id, quantity=:quantity, cost=:cost,  
ship_date=:ship_date, expected_receive_date=:expected_receive_date";
```

Update Order:

```
$query = "UPDATE" . $this->table_name. " SET supplier_id=:supplier_id,  
branch_id=:branch_id, quantity=:quantity, cost=:cost, ship_date=:ship_date,  
expected_receive_date=:expected_receive_date WHERE SKU=:SKU";
```

Delete Order:

```
$query = "DELETE FROM " . $this->table_name. " WHERE SKU=?";
```

Places_hold

Here `$table_name` = "places_hold";

Read Places_hold:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Places_hold:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE customer_id=? LIMIT 0,1";
```

Create Places_hold:

```
$query = "INSERT INTO " . $this->table_name. " SET customer_id=:customer_id,  
SKU=:SKU, quantity=:quantity, date_placed=:date_placed,  
date_released=:date_released, price=:price";
```

Update Places_hold:

```
$query = "UPDATE" . $this->table_name. " SET SKU=:SKU, quantity=:quantity,  
date_placed=:date_placed, date_released=:date_released, price=:price WHERE  
customer_id=:customer_id";
```

Delete Places_hold:

```
$query = "DELETE FROM " . $this->table_name. " WHERE customer_id=?";
```

Product

Here `$table_name` = "product_type";

Read Product:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Product:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE SKU=? LIMIT 0,1";
```

Create Aisle Product:

```
$query = "INSERT INTO " . $this->table_name. " SET SKU=:SKU, type=:type,  
name=:name, description=:description, additional_info=:additional_info,  
UPC=:UPC, regular_price=:regular_price, sale_price=:sale_price,  
club_price=:club_price, quantity=:quantity";
```

Update Product:

```
$query = "UPDATE" . $this->table_name. " SET type=:type, name=:name,  
description=:description, additional_info=:additional_info, UPC=:UPC,  
regular_price=:regular_price, sale_price=:sale_price, club_price=:club_price,  
quantity=:quantity WHERE SKU=:SKU";
```

Delete Product:

```
$query = "DELETE FROM " . $this->table_name. " WHERE SKU=?";
```

Sales_associate

Here `$table_name = "sales_associate";`

Read Sales_associate:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Sales_associate:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE SID=? LIMIT 0,1";
```

Admin Authentication:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE username=@username  
AND password=@password";
```

Create Sales_associate:

```
$query = "INSERT INTO " . $this->table_name. " SET SID=:SID,first_name=:first_name,  
last_name=:last_name, username=:username, password=:password,  
section=:section";
```

Update Sales_associate:

```
$query = "UPDATE" . $this->table_name. " SET first_name=:first_name,  
last_name=:last_name, username=:username, password=:password,  
section=:section WHERE SID=:SID";
```

Delete Sales_associate:

```
$query = "DELETE FROM " . $this->table_name. " WHERE SID=?";
```

Ships_to

Here `$table_name = "ships_to";`

Read Ships_to:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Ships_to:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE customer_id=? LIMIT 0,1";
```

Create Ships_to:

```
$query = "INSERT INTO " . $this->table_name. " SET customer_id=:customer_id,  
order_id=:order_id, branch_id=:branch_id, receive_date=:receive_date";
```

Update Ships_to:

```
$query = "INSERT INTO " . $this->table_name. " SET order_id=:order_id,  
branch_id=:branch_id, receive_date=:receive_date WHERE  
customer_id=:customer_id";
```

Delete Ships_to:

```
$query = "DELETE FROM " . $this->table_name. " WHERE customer_id=?";
```

Ships_from

Here `$table_name = "ships_from";`

Read Ships_from:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Ships_from:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE customer_id=? LIMIT 0,1";
```

Create Ships_from:

```
$query = "INSERT INTO " . $this->table_name. " SET customer_id=:customer_id,  
order_id=:order_id, branch_id=:branch_id, ship_date=:ship_date";
```

Update Ships_from:

```
$query = "INSERT INTO " . $this->table_name. " SET order_id=:order_id,  
branch_id=:branch_id, ship_date=:ship_date WHERE  
customer_id=:customer_id";
```

Delete Ships_from:

```
$query = "DELETE FROM " . $this->table_name. " WHERE customer_id=?";
```

Store_branch

Here `$table_name = "store_branch";`

Read Store_branch:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Store_branch:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE branch_id=? LIMIT 0,1";
```

Create Store_branch:

```
$query = "INSERT INTO " . $this->table_name. " SET branch_id=:branch_id, email=:email,
        phone_no=:phone_no, province=:province, city=:city, street_no=:street_no";
```

Update Store_branch:

```
$query = "UPDATE" . $this->table_name. " SET email=:email, phone_no=:phone_no,
        province=:province, city=:city, street_no=:street_no WHERE
        branch_id=:branch_id";
```

Delete Store_branch:

```
$query = "DELETE FROM " . $this->table_name. " WHERE branch_id=?";
```

Supplier

Here `$table_name = "supplier";`

Read Supplier:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Supplier:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE id=? LIMIT 0,1";
```

Create Supplier:

```
$query = "INSERT INTO " . $this->table_name. " SET id=:id, name=:name,
        phone_no=:phone_no, email=:email";
```

Update Supplier

```
$query = "UPDATE" . $this->table_name. " SET name=:name, phone_no=:phone_no,
        email=:email WHERE id=:id";
```

Delete Supplier

```
$query = "DELETE FROM " . $this->table_name. " WHERE id=?";
```

Transfer_order

Here `$table_name = "transfer_order";`

Read Transfer_order:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Transfer_order;

```
$query = "SELECT * FROM " . $this->table_name. " WHERE order_id=? LIMIT 0,1";
```

Create Transfer_order:

```
$query = "INSERT INTO " . $this->table_name. " SET customer_id=:customer_id,  
order_id=:order_id, branch_id=:branch_id, date_ordered=:date_ordered,  
invoiced=:invoiced";
```

Update Transfer_order:

```
$query = "UPDATE" . $this->table_name. " SET customer_id=:customer_id,  
branch_id=:branch_id, date_ordered=:date_ordered, invoiced=:invoiced WHERE  
order_id=:order_id";
```

Delete Transfer_order:

```
$query = "DELETE FROM " . $this->table_name. " WHERE order_id=?";
```

Works_for

Here \$table_name = "works_for";

Read Works_for:

```
$query = "SELECT * FROM " . $this->table_name;
```

Read One Works_for:

```
$query = "SELECT * FROM " . $this->table_name. " WHERE SID=? LIMIT 0,1";
```

Create Works_for:

```
$query = "INSERT INTO " . $this->table_name. " SET branch_id=:branch_id, SID=:SID,  
shift=:shift";
```

Update Works_for:

```
$query = "UPDATE" . $this->table_name. " SET branch_id=:branch_id, shift=:shift  
WHERE SID=:SID";
```

Delete Works_for:

```
$query = "DELETE FROM " . $this->table_name. " WHERE SID=?";
```

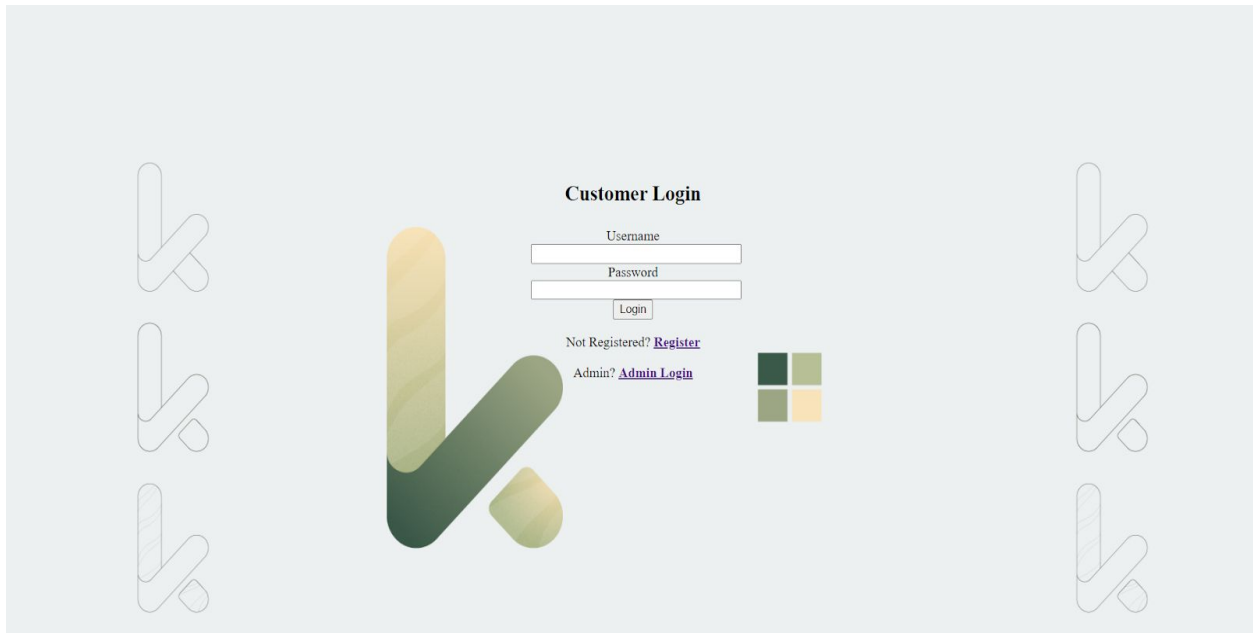
API Documentation

Refer to the attached pdf file named “KMS_TOOLS_API_Documentation.pdf” for the complete documentation of our implemented REST API for our SQL database due to its large amount of pages.

As a side note: Since we had no prior experience with API development or PHP, we used [2] as a starting point to get the basic framework of the api in php set up. The queries and classes were developed independently to communicate with our project database.

User Guide For KMS Tools Website

Customer Login Page:

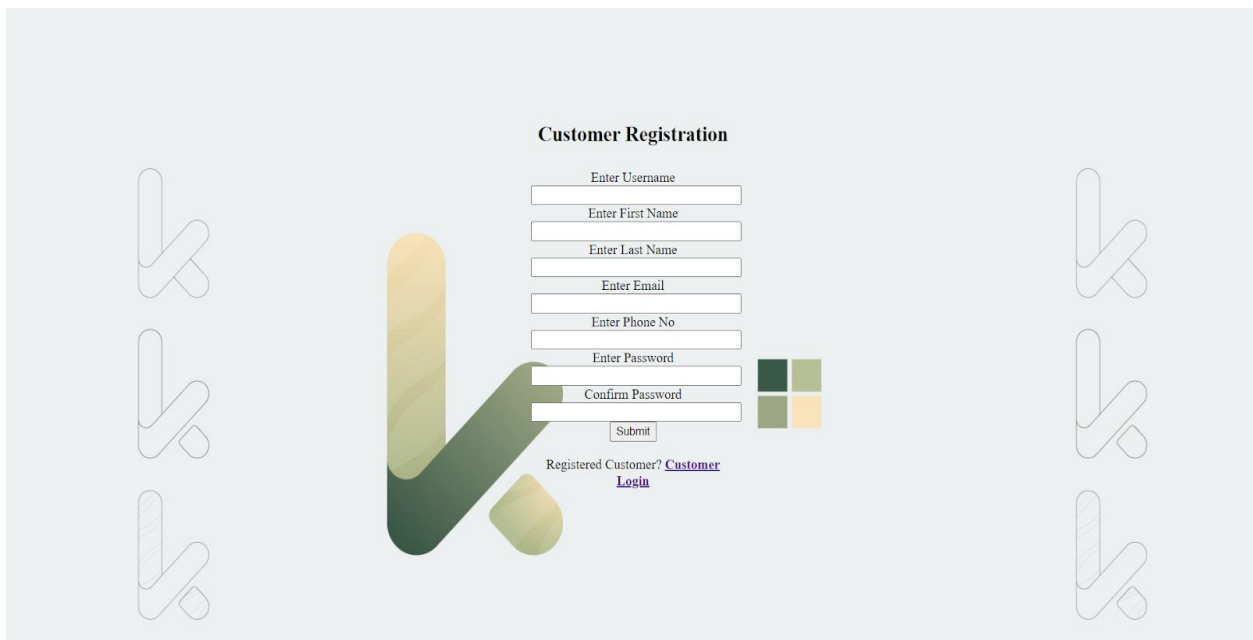


The screenshot shows the 'Customer Login' page of the KMS Tools website. The page has a light blue background with a large, stylized 'K' logo in the center, composed of yellow and green segments. On the left and right sides, there are three white 'K' logos stacked vertically. The login form is centered and includes the following elements:

- Customer Login** (Section Header)
- Username** (Text label above a text input field)
- Password** (Text label above a text input field)
- Login** (Text button)
- Not Registered? [Register](#)** (Text with a link)
- Admin? [Admin Login](#)** (Text with a link)
- A small 2x2 grid of colored squares (dark green, light green, yellow, and dark green) to the right of the form.

This login screen is available to all registered customers who would like to access the online KMS TOOLS System. This is the first page any user is directed to when using the browser based system. For customers to access the system they must log in with their username and password credentials. If the customer isn't registered in the system, they must proceed to register themselves into the system to gain access.

Registration Page:

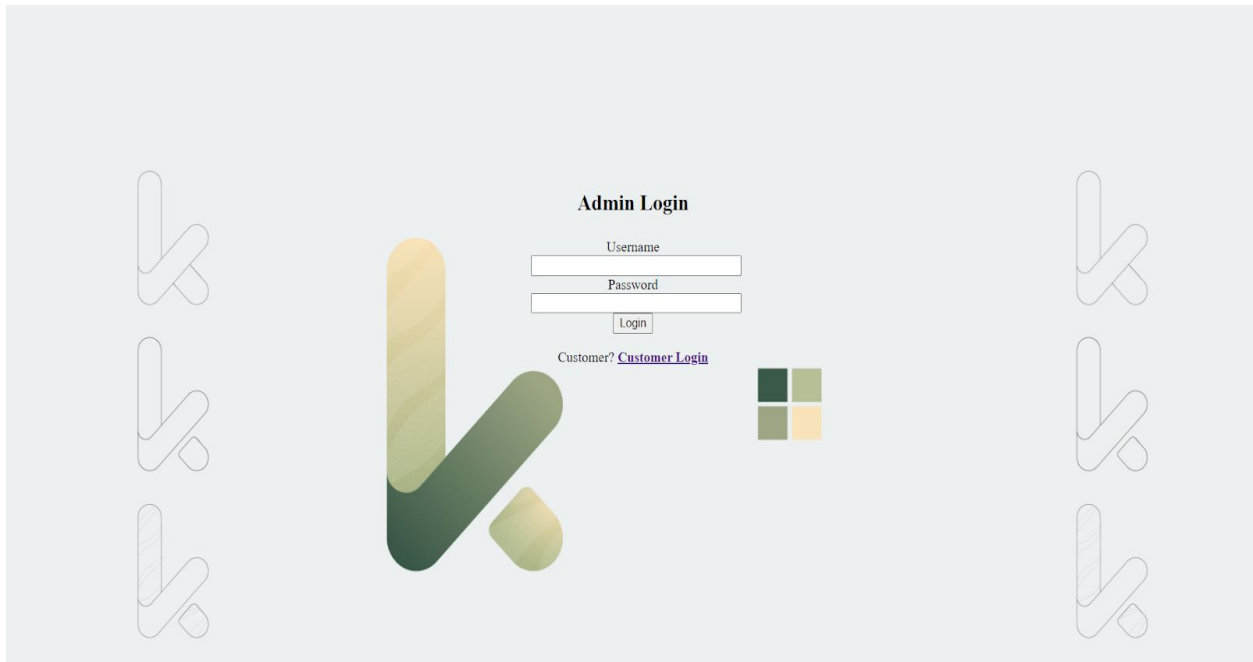


The screenshot shows the 'Customer Registration' page of the KMS Tools website. The page has a light blue background with a large, stylized 'K' logo in the center, composed of yellow and green segments. On the left and right sides, there are three white 'K' logos stacked vertically. The registration form is centered and includes the following elements:

- Customer Registration** (Section Header)
- Enter Username** (Text label above a text input field)
- Enter First Name** (Text label above a text input field)
- Enter Last Name** (Text label above a text input field)
- Enter Email** (Text label above a text input field)
- Enter Phone No** (Text label above a text input field)
- Enter Password** (Text label above a text input field)
- Confirm Password** (Text label above a text input field)
- Submit** (Text button)
- Registered Customer? [Customer Login](#)** (Text with a link)
- A small 2x2 grid of colored squares (dark green, light green, yellow, and dark green) to the right of the form.

The customer registration page allows new customers to register themselves into the system to gain access to its functionality.

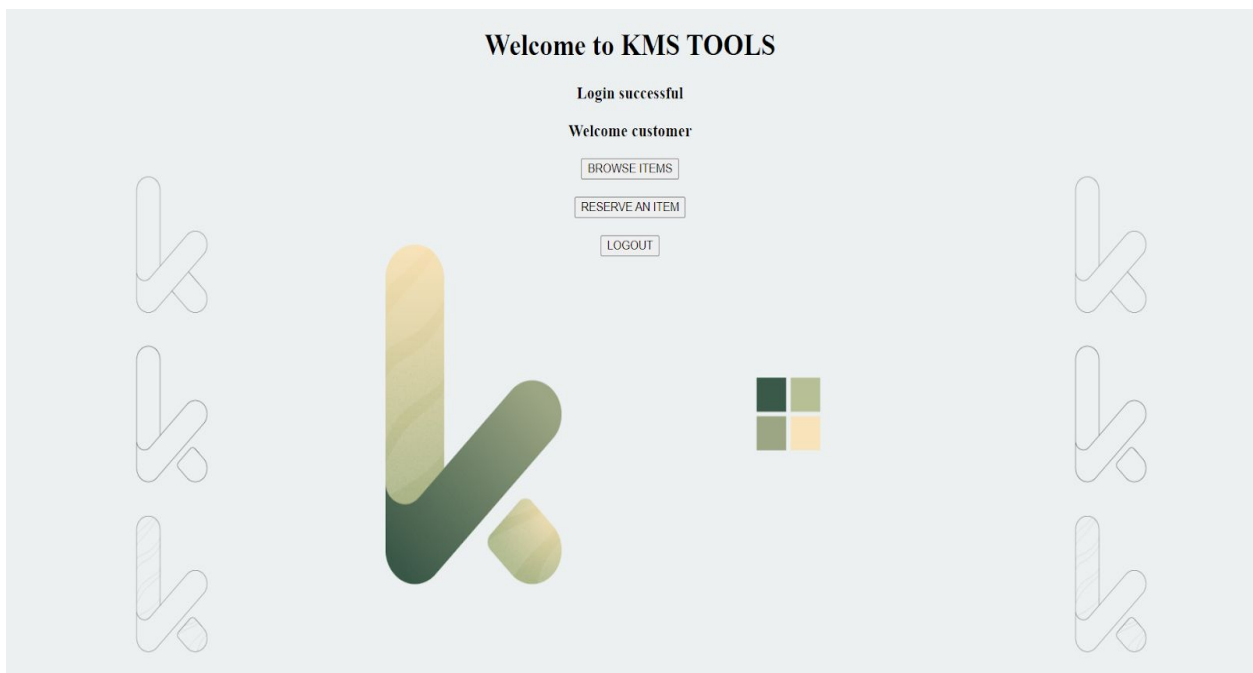
Admin Login Page:



The screenshot shows the Admin Login page. It features a central login form with the title "Admin Login". Below the title are two input fields labeled "Username" and "Password", followed by a "Login" button. Below the login button, there is a link that says "Customer? [Customer Login](#)". The page has a light blue background with a large, stylized "K" logo in the center, composed of green and yellow segments. On the left and right sides, there are vertical columns of three stylized "K" logos each. A small 2x2 grid of colored squares (dark green, light green, yellow, and dark green) is located to the right of the login button.

Admin Users/ Sales Associates have a separate login page that after verifying their credentials directs them to a separate homepage that contains admin functionalities and options.

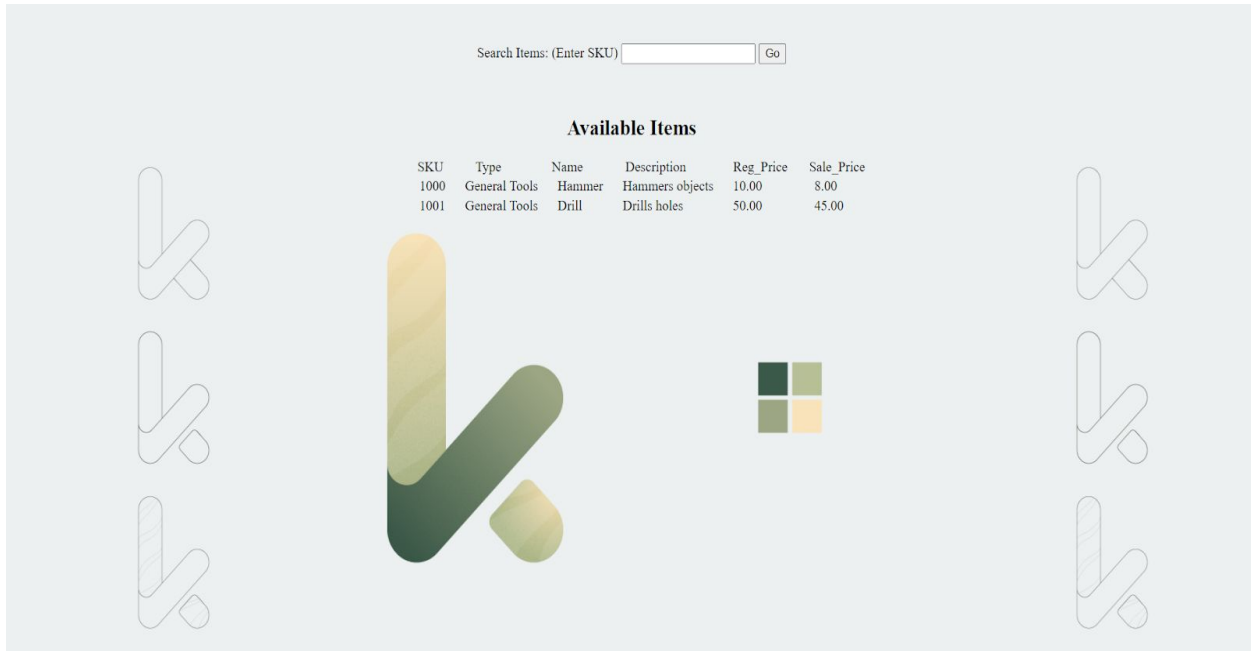
Customer Homepage:



The screenshot shows the Customer Homepage. It features a central area with the title "Welcome to KMS TOOLS". Below the title, it says "Login successful" and "Welcome customer". There are three buttons: "BROWSE ITEMS", "RESERVE AN ITEM", and "LOGOUT". The page has a light blue background with a large, stylized "K" logo in the center, composed of green and yellow segments. On the left and right sides, there are vertical columns of three stylized "K" logos each. A small 2x2 grid of colored squares (dark green, light green, yellow, and dark green) is located to the right of the buttons.

Once customers have successfully logged in, they are redirected to this page where they can access all the functionalities available to customers. Options such as Browsing products sold by KMS Tools Store Branches, reserve a product to pick up from a store branch and request a transfer of a product to a branch of choice for pickup (The last two options have yet to be implemented due to time constraints). Logout option, signs the customer off from the website and takes them back to the login screen. Note the homepage is blocked and accessible only once a customer is logged in.

Browse Items Page:




The screenshot shows the 'Browse Items' page. At the top, there is a search bar with the placeholder text 'Search Items: (Enter SKU)' and a 'Go' button. Below the search bar, the title 'Available Items' is centered. Underneath the title is a table with the following data:

SKU	Type	Name	Description	Reg. Price	Sale Price
1000	General Tools	Hammer	Hammers objects	10.00	8.00
1001	General Tools	Drill	Drills holes	50.00	45.00

The page features a light blue background with a large, stylized 'K' logo in the center, composed of green and yellow segments. The logo is surrounded by several smaller, faint 'K' logos. In the bottom right corner, there is a small 2x2 grid of colored squares (dark green, light green, yellow, and dark green).

This page displays all the available products sold by the KMS Tools Store Branches along with all their information stored in the database. The customers will also have the functionality to search for an item by entering its unique SKU value in the search box at the top of the page (this feature remains to be implemented due to time constraints). Clicking the back button on the browser redirects you back to the customer homepage to access other options.

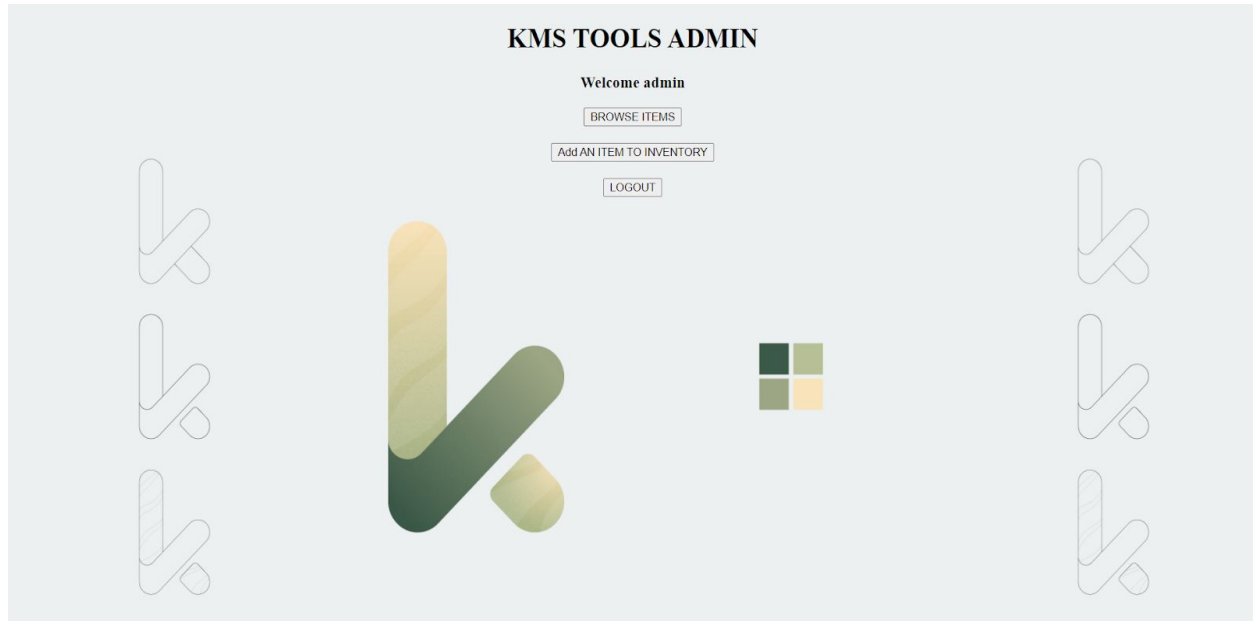
Reserve Product Page:



The screenshot shows the 'Reserve Product' page. At the top, the title 'Reserve Product' is centered. Below the title, there are two input fields: 'Product ID :' and 'Quantity :'. Below these fields are two buttons: 'Reserve?' and 'Home'. The page features a light blue background with a large, stylized 'K' logo in the center, composed of green and yellow segments. The logo is surrounded by several smaller, faint 'K' logos. In the bottom right corner, there is a small 2x2 grid of colored squares (dark green, light green, yellow, and dark green).

Through this page customers can reserve a product by entering the product SKU and its quantity to be reserved. Additionally customers will also be prompted to select their branch to see if the request can be fulfilled or not. (Note: This page isn't functional due to time constraints)

Admin Homepage:



This is the homepage an admin/ sales associate user will see once they successfully complete the login screen with valid credentials. Admin users have access to functions such as browse the items available to customers, add/ update the stock of products, order products from their suppliers and fulfill transfer order requests. These features couldn't be implemented due to time constraints. The logout button logs the admin user off the system and redirects them back to the admin login page. Note the admin homepage is not accessible unless login has been completed successfully.

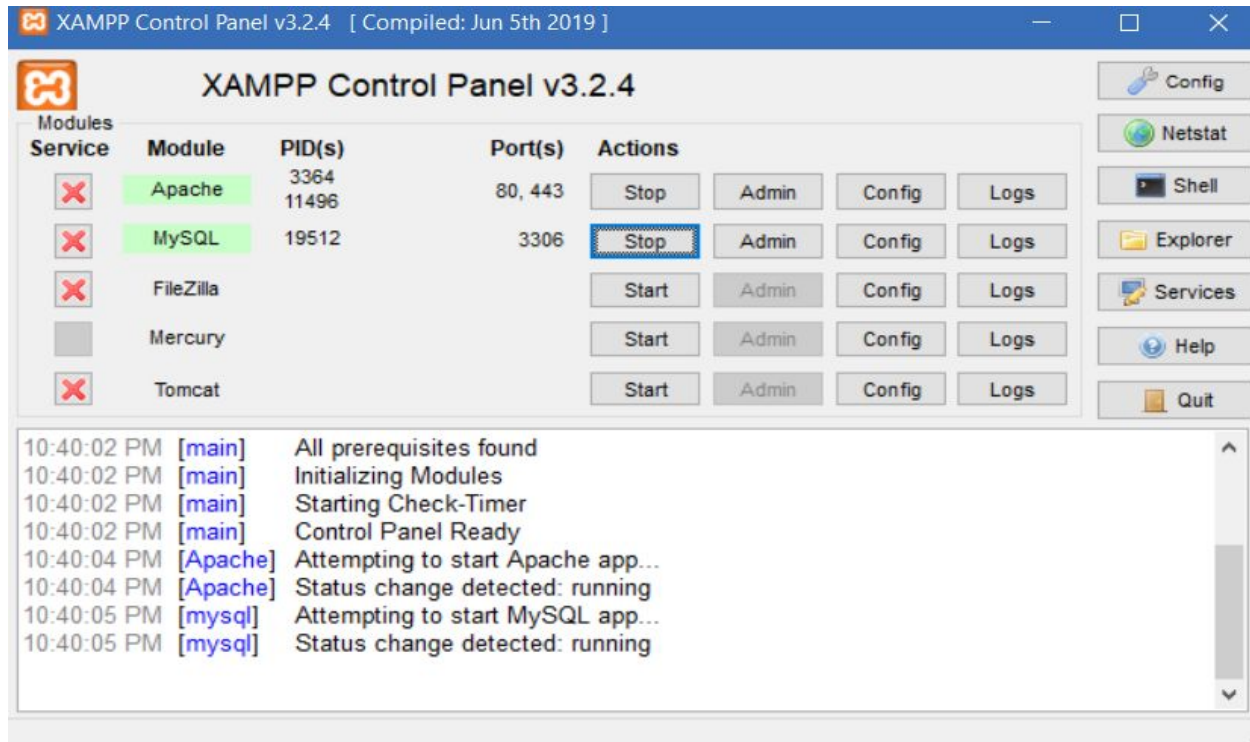
Add Product To Inventory Page:

The screenshot shows the 'Add an Item' form. It has four input fields: 'Item Name' with the value 'wrench', 'Type' with the value 'tool', 'Quantity Available' with the value '0', and 'Regular Price' with the value '0.00'. Below these fields is an 'Add Item' button. The background is light blue with the same stylized 'K' logo and decorative elements as the admin homepage.

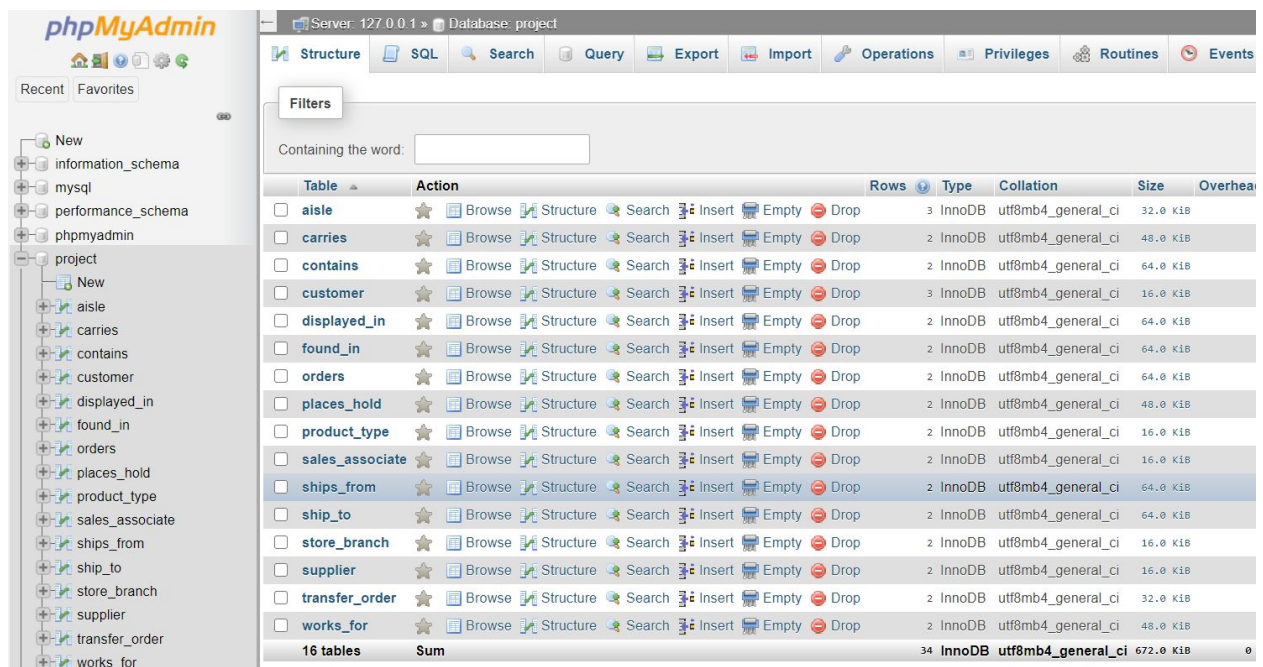
Through this page admin users can add product stock available to customers to purchase by entering its details. Page needs implementation but is designed to update, create, remove products accessible to customers. Further items will be able to be added to specific branches as they become available. The back button of the browser redirects back to the admin homepage to access other options.

User Guide To Run Website/ API for testing

1. Download XAMPP Server Host/ similar application to run a local apache and SQL server.
2. Have MYSQL also installed on the system.
3. Download the folders named “phprest” and “KMSTOOLS”
4. Place these folders in the htdocs folder/ root folder of XAMPP Server host software or tool of choice capable of running apache.
5. Download the “project.sql” database and import it to the PHPMYADMIN database list through <http://localhost/phpmyadmin/index.php> page link if u have xampp installed on default settings. If you're using 'xampp' for example, then create a new database named project in 'phpmyadmin' and import the 'project.sql' file to it. Make sure to uncheck the 'Enable foreign key checks' and 'Allow the interruption of an import in case the script detects it is close to the PHP timeout limit' options.
6. Once you can access the database and both apache and sql is running from XAMPP/ similar software, to use the website enter this url to your browser:
<http://localhost/KMSTools/login.php>
7. To test the api endpoints using postman, u can call the requests with
<http://localhost/phprest/api/> endpoint you want to call.



Example of XAMPP running the apache and MYSQL services



Example of phpMyAdmin page with the project.sql database imported

References

- [1] Canadian Retail Sales Growth Lowest in a Decade: Expert. (2020, February 27). Retrieved December 11, 2020, from <https://www.retail-insider.com/retail-insider/2020/02/canadian-retail-sales-growth-lowest-in-a-decade-expert/>
- [2] Dalisay, M. (2020, June 28). How To Create A Simple REST API in PHP - Step By Step Guide Retrieved December 1, 2020, from <https://codeofaninja.com/2017/02/create-simple-rest-api-in-php.html>