Warehouse Management System

Database Management

CMPT 308N-113

Bezos' Indentured Servants

Marist College
School of Computer Science and Mathematics

Submitted To:
Dr. Reza Sadeghi

9/6/2023

Table of Contents

Table of Figures

Table of Tables

I. Project Description of *Bezos' Indentured Servants*

Team Name

Name of the Team ----------*Bezos' Indentured Servants*

Github Link

https://github.com/gabrielle-knapp1/Project-1-Database-Man-

Team Members

1. Ethan Morton ---------- ethan.morton1@marist.edu (Team Head)

2. Gabrielle Knapp----------gabrielle.knapp1@marist.edu (Team Member)

Description of Team Members

1. ***Ethan Morton***

   I am a sophomore and I'm majoring in computer science with a concentration in software development. I am minoring in mathematics, information technology, information systems, and cybersecurity. I know C#, C++, Java, Python, HTML, CSS, JavaScript, and SQL. I am from Granby, Connecticut and I enjoy playing tennis, hanging out with my friends, and playing games. Some clubs I'm involved in are campus ministry, computer society, games society, and club tennis. Together we chose me to be the Team Head because of my ability to manage communications.

2. ***Gabrielle Knapp***

   Gabrielle Knapp is a sophomore at Marist College majoring in Computer Science with minors in Spanish and Economics. She comes from Carlisle, Pennsylvania and has been enjoying her time adjusting to college life. Activities she currently is participating in include intramural badminton and the games society. In her free time, she loves reading, going on

long walks, and playing board games. She is excited to see how she learns and grows her programming skills throughout this class and her entire time at Marist. Gabrielle is excited to bring her hardworking, can-do attitude to her group in this project for her Database Management class. She chose to work with Ethan on this project because she has enjoyed working with him in other classes, including working together on their project in Dr. Sadeghi's Intro to Programming class. Together, as a pair, they chose Ethan to be the Team Head because of his excellent ability to manage communications.

II. Project Objectives

Summary

We have selected project sample 4, the warehouse management system. The warehouse management system (WMS) provides an organized way of storing different products and elements in a warehouse. You can consider a library as a warehouse, which maintains books' details and user libraries. A general WMS stores details of name and identification number of products, their store time, the required storage condition, price, weight, height, etc. Following this, this system allows guest users to search for different content and request to borrow/buy them. Our WMS will store the data of different user types in distinct SQL tables.

Modules

- Admin Roles
    - Admin has login (username/password) and ability to change that login
    - Admin can remove users from WMS
    - Admin has ability to add a guest user with a login (guest has limited abilities)
        - Guest user cannot define/remove other users
    - Admin can add, delete, and edit items to WMS with various details
    - Admin can view, accept, and reject the list of borrowing requests
- User Roles
    - Users can search through items in WMS depending on various item details
    - Users have ability to save favorite items
    - Users can request to borrow/buy specific items at specific times
    - Users have ability to view the history of borrowed/bought items
- WMS should be user-friendly software

    - Welcome page
    - Menu with all functions
    - All functions in a tabular format
    - Well-organized list of requested items
    - Exit function with friendly goodbye
    - Should show warnings IF:
        - The Admin user tries to add a new item to the library with an existing ID
        - If a guest user tries to borrow more than 3 items
        - A user search request returns null items
- WMS should protect User's information/data
    - WMS passwords & recorded info should be Ciphered using Caesar Cipher Method

III. Review the Related Work

1) BR Williams Warehouse Management System (1)

a) Positive Aspects

i) Inventory history and transaction logs. Makes it easy for users to work with the system.

b) Negative Aspects

i) Uses barcodes to scan which automatically update the database with the correct information.

2) Koha Management System (2)

a) Positive Aspects

i) Intuitive navigation for users. This also has very good GUI and user graphics.

b) Negative Aspects

i) Built for library management rather than warehouse management.

3) ShipHero Warehouse Management System (3)

a) Positive Aspects

i) Users can return items. It also is very well organized.

b) Negative Aspects

i) Multi-carrier shopping: searching the same items from different stores won't work in our project.

IV. Merits of Project

Merits:

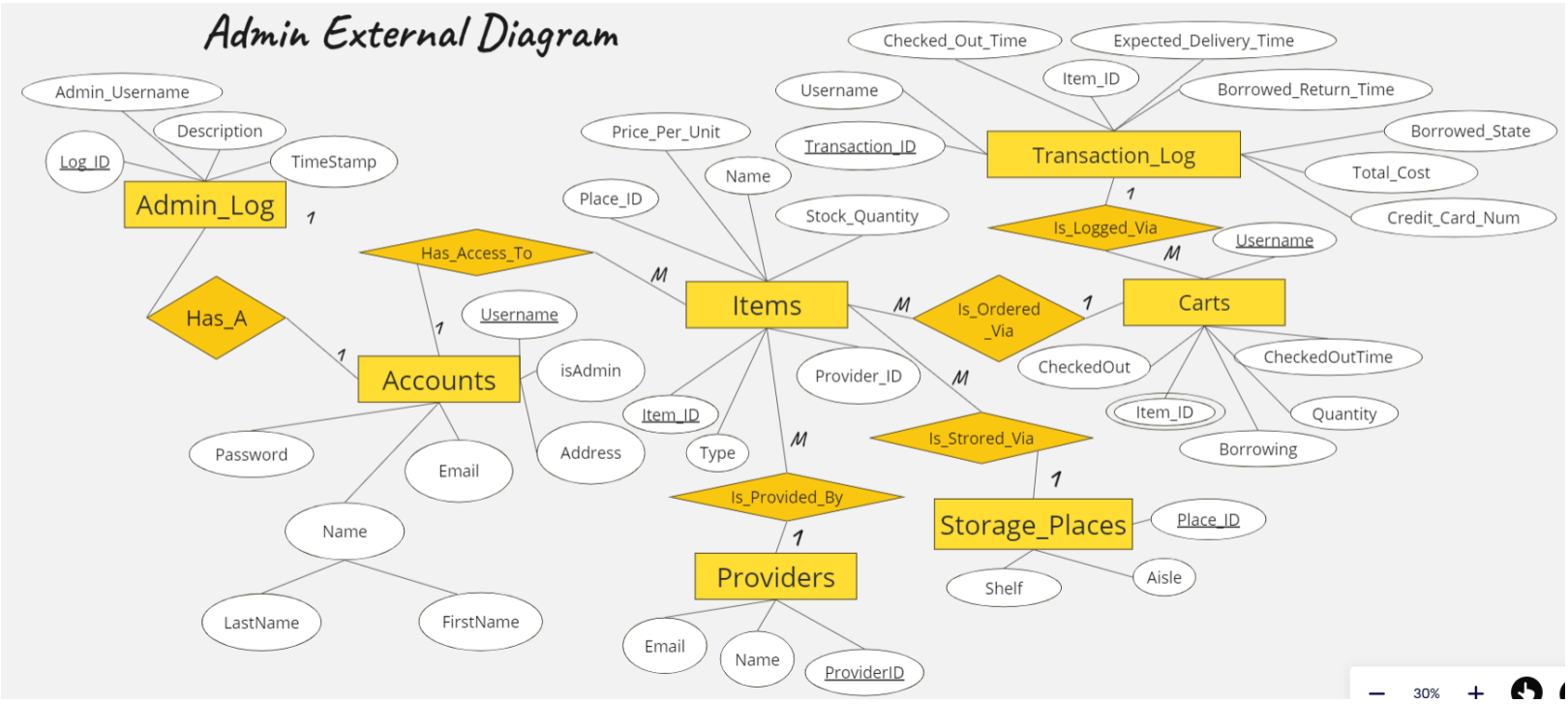
- The WMS will organize all of the items in the warehouse and will allow users to easily access a list of all of these items and their details. This is something that we learned after studying other Warehouse Management Systems and realizing the benefits of including this practice.
- We plan on implementing the positives of other Warehouse Management Systems, including the implementation of detailed inventory logs, allowing for clear details about returning items, and creating an intuitive navigation for users.
- The WMS will clearly differentiate the roles of User and Admin, and allow Admin complete control over the warehouse and the privileges of the users. This is according to the necessary components detailed in the rubric.

V. Entity Relationship Models & Diagrams

Admin External Model:

The administrators require the accounts feature, the admin_log feature, the items table, the providers table, the storage_places table, the carts table, and the transaction Log

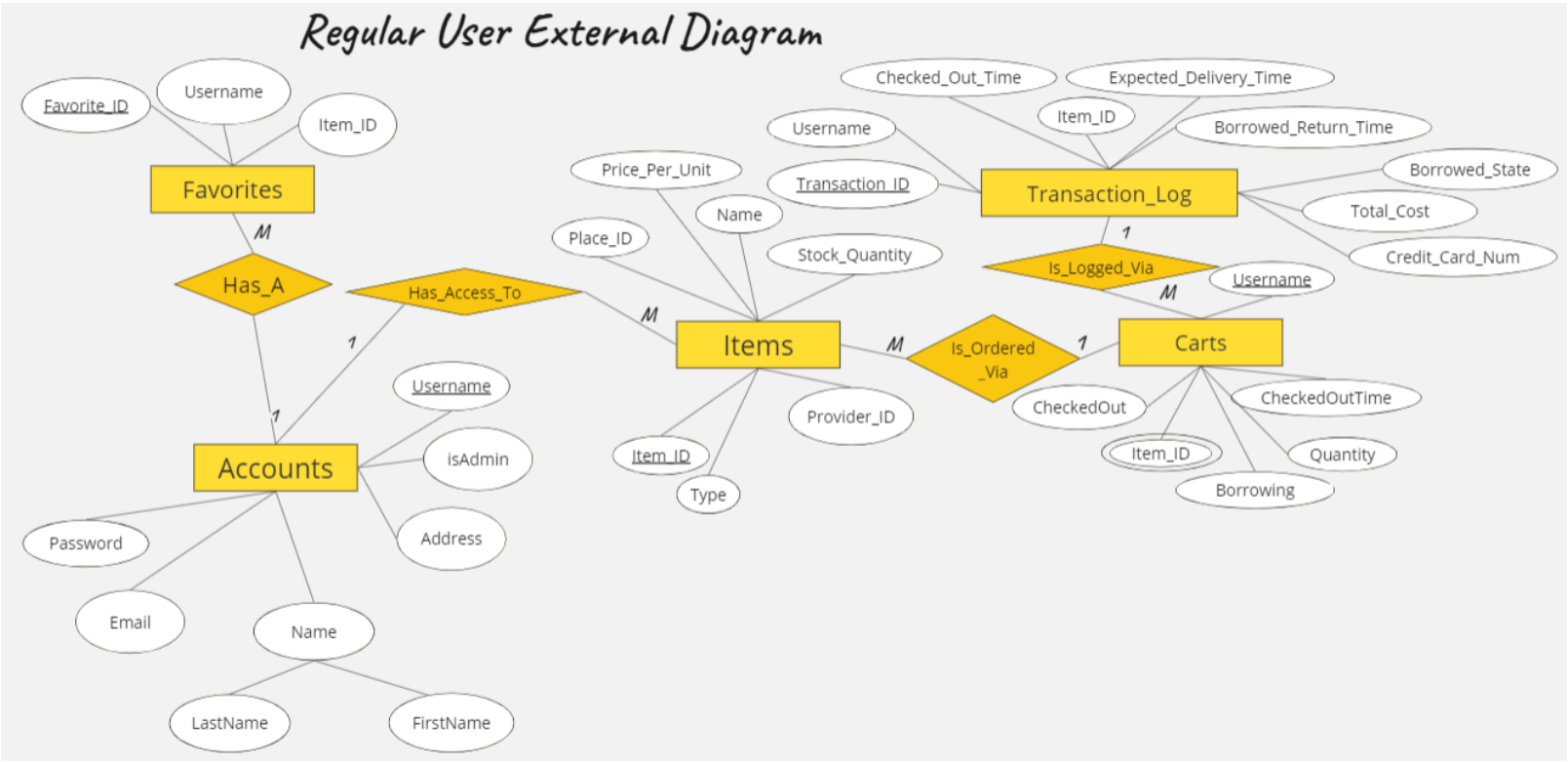
Figure 1: Admin External Diagram:



Regular User External Model:

The regular users require the favorites table, the accounts table, the items table, the carts table, and the transaction_log table.

Figure 2: Regular User External Diagram:



Overview (ER Model):

- Here, we have a more detailed version of what we might need for the warehouse and what is described in the EER Diagram. We have the Account, which is anyone using the WMS. The Account is either Admin or not admin, and that is controlled by the isAdmin boolean. If they are admin, then the user has access to the Admin abilities, which are listed in the Admin_Log table. Each account also has the ability to have a list of their favorite items, stored in the Favorites table. The items which could be included in this table are all stored in the Items table, which stores all of the items in the warehouse along with all of the important information about those items. The Providers table shows the list of providers who provide the items and their important info. The Items storage information is stored in the Storage_Places table, which lists its location in the warehouse. The items can be requested for purchase or requested to be borrowed by accounts and that is stored in the Carts table. A history of previous orders is stored in the Transaction_Log table.

Description of each entity, attribute, relationship, participation, and cardinality:

- Accounts
    - Username: ID of the user
    - Email & Address: Primary contact information of this user

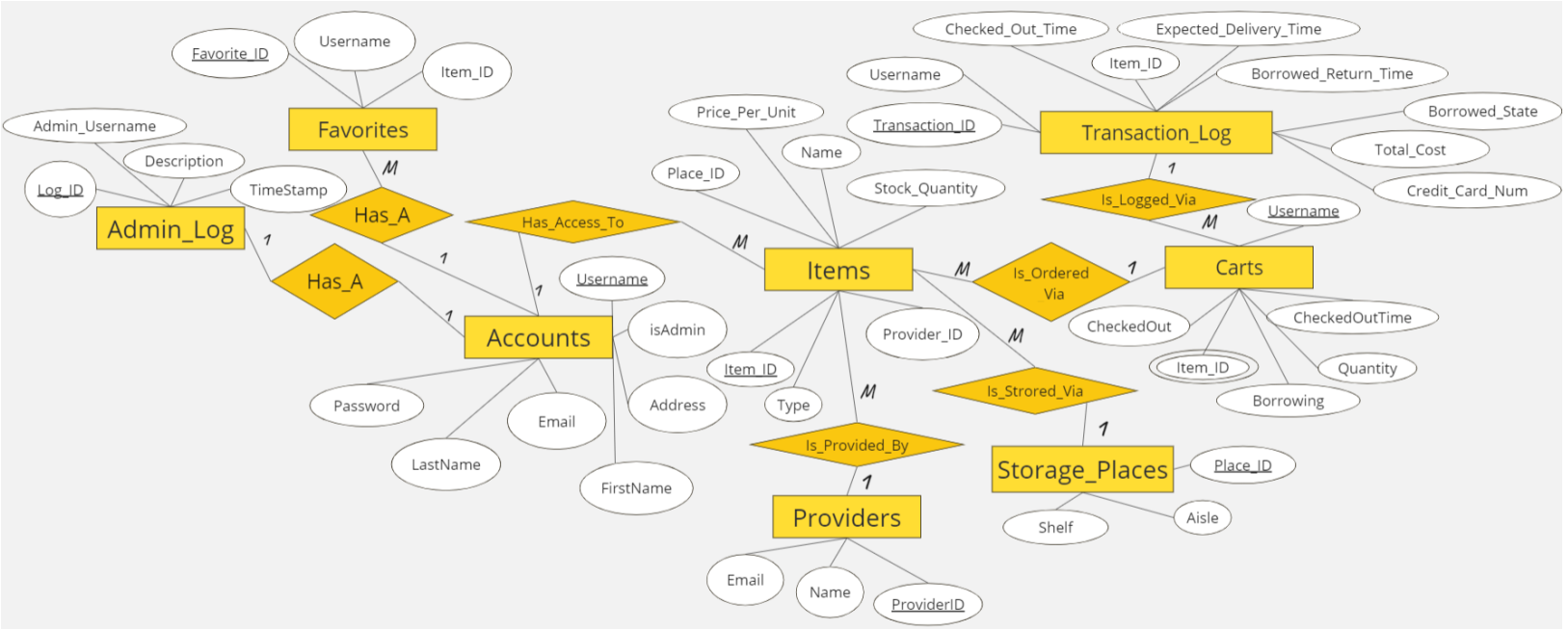
    - Password: login information
    - FirstName & LastName: name of account holder
    - isAdmin: type of user
- Providers
    - ProviderID: ID of provider
    - Email & Name: Contact Info
- Admin_Log
    - Log_ID: Admin ID of administrator
    - Admin_Username: Username of this administrator
    - Description: Action taken
    - Timestamp: When this action was taken
- Favorites
    - FavoriteID: ID of this favorite Item
    - Username: ID of user who has this as their favorite
    - Item_ID: ID of this item
- Items
    - PlaceID: ID of where item is stored
    - ItemID: ID of this particular item
    - Price_Per_Unit: Price of 1 item
    - StockQuantity: How much item is stored in warehouse
    - Name: Title of item
    - ProviderID: ID of provider
    - Type: Category of Item
- Storage_Places
    - PlaceID: ID of the storage location of this item
    - Aisle & Shelf: Specific location where this item is stored
- Carts
    - AcceptedYN: Boolean, Whether this order has been accepted
    - ItemID: ID of the item being requested
    - Username: ID of the user
    - CheckedOut: Whether the item is checked out
    - Borrowing: Whether the item is being borrowed
    - CheckedOutTime: timestamp of request
    - Quantity: quantity of items requested
- Transaction_Log
    - Total_Cost Total of price
    - ItemID: ID of the item being ordered
    - Username: IDs of the user who placed these orders
    - TransactionID: ID of the transaction log (PK)
    - CreditCardNum: Credit Card Number of the account

    -
- Item → Has A → Provider
    - Each item has a provider
- Account → Has A → AdminLog
    - Each user has a value declaring whether it has access to Admin powers
- Account → Has A → FavoriteItem
    - Each user has access to a list of favorite items
- Account → Has Access to → Items
    - Each user has access to the table of items
- Item → Is Stored Via → Storage_Places
    - Each item has a PlaceID with info on where it is stored
- Item → Is OrderedVia→ Carts
    - Each Item can be ordered via an Order Request in the Cart
- Carts → Are logged Via →Transaction Log
    - Each Order Request ever requested can be viewed in the TransactionLog table

Detailed Description of Creation Process

In creating this mini-world for a warehouse management system, I carefully selected and designed the entities, attributes, relationships, participations, and cardinalities to represent the key elements of the system. I chose entities like "Accounts," "Providers," "Items," and "Carts" to reflect the fundamental components within the warehouse environment. Within each entity, I defined attributes such as "Username," "Email," and "Price_Per_Unit" to capture essential properties and information. I established relationships like "Item → Has A → Provider" and "Account → Has A → AdminLog" to connect these entities, mirroring the real-world associations between items and providers, as well as between user accounts and administrative logs. I carefully considered the participation of entities in relationships and defined cardinalities (e.g., one-to-many or many-to-many) to model the interactions and dependencies in the system effectively. This conceptual model serves as the foundation for building a database that can manage inventory, user accounts, orders, and more within a warehouse environment.

Figure 3 (ER Diagram):



VI. Enhanced Entity Relationship Model (EER Model)

Description about keys & relationships:

- Each Account:
    - has a unique username
    - has a password
    - is either an admin or guest
    - has a first and last name
    - has an address to ship the items to
    - has an email to contact the user
- Each Item:
    - has a unique item ID
    - has a type
    - has a name
    - has a provider
    - has a quantity of that item in the warehouse to prevent redundancy
    - has a place it's stored in the warehouse
    - has a price per unit
- Each Transaction Log:
    - has a unique transaction ID
    - references all the items in each cart via a username
    - has the time of checkout
    - has an expected delivery time
    - has an actual delivery time
    - has a time when borrowed items should be returned
    - has a state for items being borrowed
    - has the total cost
    - has the credit card number used for the transaction
- Each Favorite:
    - has a unique favorite ID
    - holds the username of the account that has favorited the item
    - has the item being favorited
- Each Storage Place:
    - has a unique ID
    - has an aisle number

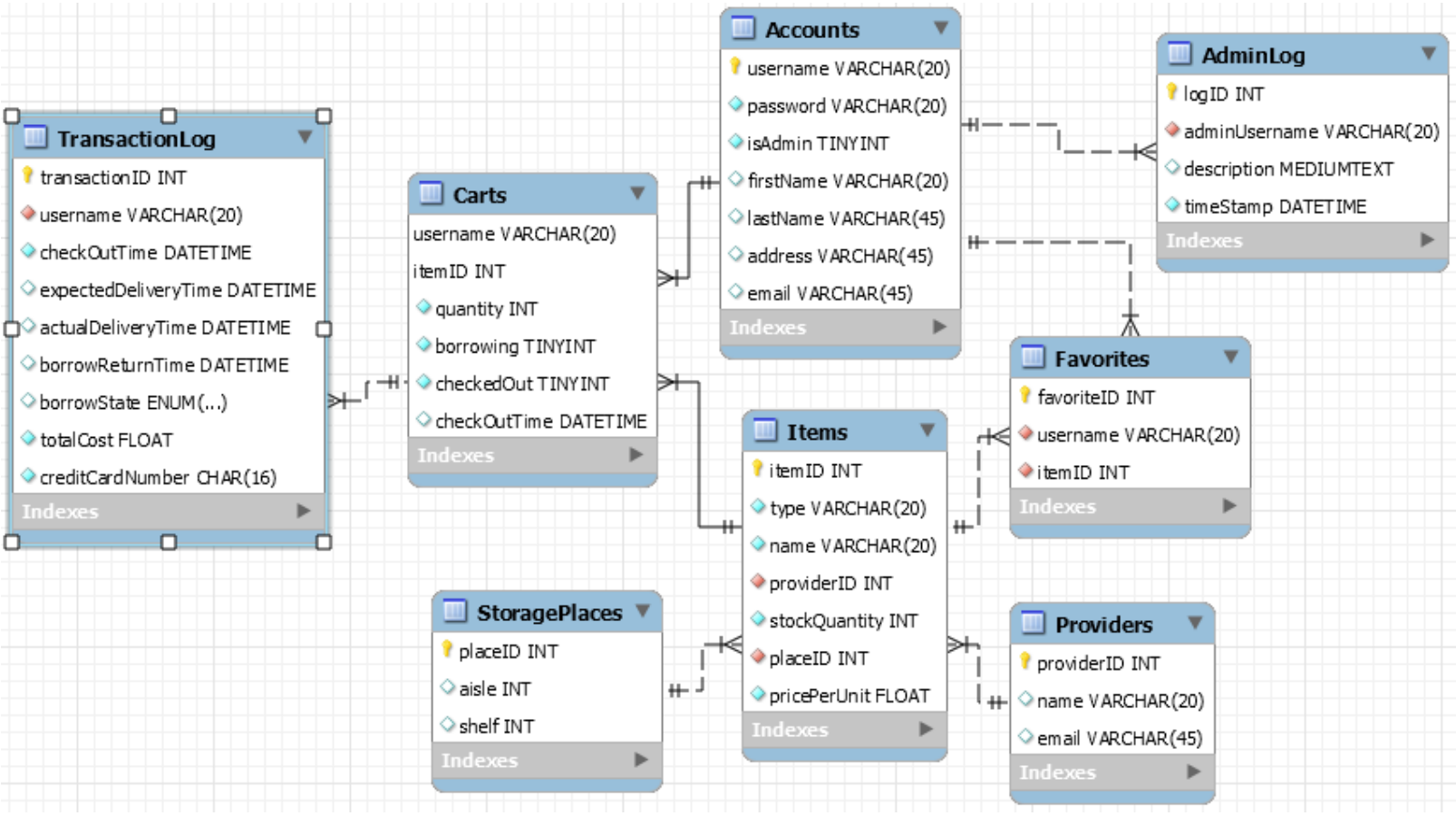
    - has a shelf number
- Each Provider:
    - has a unique ID
    - has a name
    - has an email
- Each Admin Log:
    - has a unique ID
    - has the username of the admin who did an action
    - has a description of the action
    - has a timestamp
- Each Cart:
    - has a unique username-itemID pair
    - has a checkout time
    - has the quantity of that item
    - stores whether the item is being borrowed
    - stores whether the cart has been checked out

Description of implementation of these features:

- Account
    - Logging in will give you access to the website and only show your personal data.
    - Admins may remove specific accounts.
    - Admins may approve/reject borrow requests
- Admin Log
    - Admins may approve or deny borrow requests.
    - Admins may ban users from using the WMS.
    - All admin-related actions are logged here.
- Items
    - All items in the warehouse will be displayed.
    - Users can search for items and sort the list.
    - Admins can edit the list of items.
- Transaction Log
    - Users will be able to view their past transactions.
    - Users will be able to make transactions by checking out the cart.
    - Admins can view all transactions.
- Favorites

    - Each user can view the items they favorited to access them quickly.
- Carts
    - Users can have 1 current cart which stores the items they'd like to buy but have not paid for.
    - When users pay for the items in their cart, the cart is saved to the database and they are given a new empty cart.
    - Carts that have checked out are referenced by the transaction log so users can view their previous purchases.

Figure 2 (EER Diagram):



VII. Database Development

SQL Code:

```
create database Warehouse;
use Warehouse;

create table Accounts(
        username varchar(20) primary key,
    password varchar(20) not null,
    isAdmin bool not null default false,
    firstName varchar(20),
    lastName varchar(20),
    address varchar(45),
    email varchar(45)
);

create table AdminLog(
        logID int primary key,
    adminUsername varchar(20),
    description text,
    timeStamp datetime not null,
    foreign key (adminUsername) references Accounts(username)
);

create table Carts(
        username varchar(20) primary key,
    itemID int primary key,
    quantity int not null default 0,
    borrowing bool not null default false,
    checkedOut bool not null default false,
    foreign key (username) references Accounts(username),
    foreign key (itemID) references Items(itemID)
);

create table Favorites(
```

```
        favoriteID int primary key,
    username varchar(20),
    itemID int,
    foreign key (username) references Accounts(username),
    foreign key (itemID) references Items(itemID)
);

create table Items(
        itemID int primary key,
    type varchar(20) not null,
    name varchar(20) not null,
    providerID int,
    stockQuantity int not null default 0,
    placeID int,
    pricePerUnit float not null default 0.00,
    foreign key (providerID) references Providers(providerID),
    foreign key (placeID) references StoragePlaces(placeID)
);

create table Providers(
        providerID int primary key,
    name varchar(20),
    email varchar(45)
);

create table StoragePlaces(
        placeID int primary key,
    aisle int,
    shelf int
);

create table TransactionLog(
        transactionID int primary key,
    username varchar(20),
    checkoutTime datetime not null,
```

```
    expectedDeliveryTime datetime,
    actualDeliveryTime datetime,
    borrowReturnTime datetime,
    borrowState enum("pending", "accepted", "rejected"),
    totalCost float not null default 0.00,
    creditCardNum char(16) not null,
    foreign key (username) references Carts(username)
);
```

Table 1: Tabular Description of ER Diagram + SQL Code

ER Entity	SQL Table	Attributes	Relationships
Accounts	Accounts	username (PK), password, isAdmin, firstName, LastName, Address, Email	
AdminLog	AdminLog	logID (PK), adminUsername, description, timeStamp	adminUsername is a foreign key referencing Accounts(username)
Carts	Carts	username (PK), itemID (PK), quantity, borrowing, checkedOut	username is a foreign key referencing Accounts(username) itemID is a foreign key referencing Items(itemID)
Favorites	Favorites	favoriteID (PK), username, itemID	username is a foreign key referencing Accounts(username) itemID is a foreign key referencing Items(itemID)
Items	Items	itemID (PK), type, name, providerID,	providerID is a foreign key referencing Providers(providerID) placeID is a foreign key

		stockQuantity, placeID, pricePerUnit	referencing StoragePlaces(placeID)
Providers	Providers	providerID (PK), name, email	
Storage Places	Storage Places	placeID (PK), aisle, shelf	
Transaction Log	Transaction Log	transactionID (PK), username, checkoutTime, expectedDeliveryTime, actualDeliveryTime, borrowReturnTime, borrowState, totalCost, CreditCardNum	username is a foreign key referencing Carts(username)

VIII. Loading Data and Performance Enhancements

Handling Foreign Key Constraints:

We faced an insertion error due to foreign key constraints but we were able to address this by temporarily turning off the checking process of foreign keys during data insertion. Also, foreign keys that reference keys from other tables are given the constraints: on update cascade, on delete restrict.

Importing Data

When importing the data, our priority was to test all of the range of our featured data types. For example, we strived to import maximum and minimum values as well as various boolean varchar, and int values. We faced an insertion error due to foreign key constraints but we were able to address this by temporarily turning off the checking process of foreign keys during data insertion.

Importing Data Code:

```
-- Insert sample data into the Accounts table
INSERT INTO Accounts (username, password, isAdmin, firstName, lastName, address, email)
VALUES
    ('user1', 'pass1', false, 'John', 'Doe', '123 Main St', 'john.doe@email.com'),
    ('user2', 'pass2', false, 'Jane', 'Smith', '456 Elm St', 'jane.smith@email.com'),
    ('admin1', 'adminpass', true, 'Admin', 'User', '789 Oak St', 'admin@email.com'),
    ('user3', 'pass3', false, 'Alice', 'Johnson', '101 Maple St', 'alice@email.com'),
    ('user4', 'pass4', false, 'Bob', 'Brown', '555 Pine St', 'bob@email.com'),
    ('user5', 'pass5', false, 'Eve', 'White', '777 Birch St', 'eve@email.com'),
    ('user6', 'pass6', false, 'Charlie', 'Green', '999 Cedar St', 'charlie@email.com'),
    ('user7', 'pass7', false, 'David', 'Lee', '111 Walnut St', 'david@email.com'),
    ('user8', 'pass8', false, 'Grace', 'Taylor', '222 Oak St', 'grace@email.com'),
    ('user9', 'pass9', false, 'Sam', 'Miller', '333 Maple St', 'sam@email.com');
```

```
-- Insert sample data into the Providers table
INSERT INTO Providers (providerID, name, email)
VALUES
    (1, 'Provider1', 'provider1@email.com'),
    (2, 'Provider2', 'provider2@email.com'),
    (3, 'Provider3', 'provider3@email.com'),
    (4, 'Provider4', 'provider4@email.com'),
    (5, 'Provider5', 'provider5@email.com'),
    (6, 'Provider6', 'provider6@email.com'),
    (7, 'Provider7', 'provider7@email.com'),
    (8, 'Provider8', 'provider8@email.com'),
    (9, 'Provider9', 'provider9@email.com'),
    (10, 'Provider10', 'provider10@email.com');

-- Insert sample data into the StoragePlaces table
INSERT INTO StoragePlaces (placeID, aisle, shelf)
VALUES
    (1, 1, 1),
    (2, 1, 2),
    (3, 2, 1),
    (4, 2, 2),
    (5, 3, 1),
    (6, 3, 2),
    (7, 4, 1),
    (8, 4, 2),
    (9, 5, 1),
    (10, 5, 2);

-- Insert sample data into the Items table
INSERT INTO Items (itemID, type, name, providerID, stockQuantity, placeID, pricePerUnit)
VALUES
```

```
    (1, 'Electronics', 'Laptop', 1, 50, 1, 799.99),
    (2, 'Electronics', 'Smartphone', 2, 100, 2, 499.99),
    (3, 'Clothing', 'T-Shirt', 3, 200, 3, 19.99),
    (4, 'Clothing', 'Jeans', 4, 150, 4, 39.99),
    (5, 'Groceries', 'Cereal', 5, 300, 5, 3.99),
    (6, 'Groceries', 'Milk', 6, 500, 6, 2.49),
    (7, 'Books', 'Novel', 7, 30, 7, 12.99),
    (8, 'Books', 'Textbook', 8, 20, 8, 79.99),
    (9, 'Furniture', 'Sofa', 9, 10, 9, 599.99),
    (10, 'Furniture', 'Table', 10, 15, 10, 199.99);

-- Insert sample data into the AdminLog table
INSERT INTO AdminLog (logID, adminUsername, description, timeStamp)
VALUES
    (1, 'admin1', 'Admin logged in', NOW()),
    (2, 'admin1', 'Added new provider', NOW()),
    (3, 'admin1', 'Updated user information', NOW()),
    (4, 'admin1', 'Deleted an item', NOW()),
    (5, 'admin1', 'Logged out', NOW()),
    (6, 'admin1', 'Admin logged in', NOW()),
    (7, 'admin1', 'Added new item', NOW()),
    (8, 'admin1', 'Changed password policy', NOW()),
    (9, 'admin1', 'Performed backup', NOW()),
    (10, 'admin1', 'Admin logged out', NOW());

-- Insert sample data into the Carts table
INSERT INTO Carts (username, itemID, checkOutTime, quantity, borrowing, checkedOut)
VALUES
    ('user1', 1, NOW(), 1, false, false),
    ('user2', 2, NOW(), 2, false, false),
    ('user3', 3, NOW(), 1, true, false),
```

```
    ('user4', 4, NOW(), 3, false, true),
    ('user5', 5, NOW(), 2, false, false),
    ('user6', 6, NOW(), 1, false, false),
    ('user7', 7, NOW(), 4, true, false),
    ('user8', 8, NOW(), 1, false, true),
    ('user9', 9, NOW(), 2, false, false),
    ('user10', 10, NOW(), 1, false, false);

-- Insert sample data into the Favorites table
INSERT INTO Favorites (favoriteID, username, itemID)
VALUES
    (1, 'user1', 3),
    (2, 'user1', 5),
    (3, 'user2', 1),
    (4, 'user3', 2),
    (5, 'user3', 4),
    (6, 'user4', 6),
    (7, 'user5', 8),
    (8, 'user6', 10),
    (9, 'user7', 9),
    (10, 'user8', 7);

-- Insert sample data into the TransactionLog table
INSERT INTO TransactionLog (transactionID, username, checkoutTime, expectedDeliveryTime, actualDeliveryTime,
borrowReturnTime, borrowState, totalCost, creditCardNum)
VALUES
    (1, 'user1', NOW(), NOW(), NOW(), NOW(), 'pending', 39.97, '1234567812345678'),
    (2, 'user2', NOW(), NOW(), NOW(), NOW(), 'accepted', 999.99, '9876543298765432'),
    (3, 'user3', NOW(), NOW(), NOW(), NOW(), 'rejected', 49.95, '1111222233334444'),
    (4, 'user4', NOW(), NOW(), NOW(), NOW(), 'pending', 99.99, '5555666677778888'),
    (5, 'user5', NOW(), NOW(), NOW(), NOW(), 'accepted', 199.95, '1234567890123456'),
```

```
    (6, 'user6', NOW(), NOW(), NOW(), NOW(), 'rejected', 79.99, '9876543210987654'),
    (7, 'user7', NOW(), NOW(), NOW(), NOW(), 'pending', 159.99, '1111222233445566'),
    (8, 'user8', NOW(), NOW(), NOW(), NOW(), 'accepted', 49.95, '8888777766665555'),
    (9, 'user9', NOW(), NOW(), NOW(), NOW(), 'rejected', 29.99, '4444333322221111'),
    (10, 'user10', NOW(), NOW(), NOW(), NOW(), 'pending', 79.99, '2222333344445555');
#Here I turn back on the checking process of foreign keys
SET FOREIGN_KEY_CHECKS=0;
```

Insertion Optimization

When inserting data into the database, it is not optimal to open a connection, insert some data, close the connection, reopen the connection, insert some more data, then close the connection again. It is much more efficient to insert all the data over one connection interval. Here are the specifics:

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting row: (1 × size of row)
- Inserting indexes: (1 × number of indexes)
- Closing: (1)

It takes longer to set up the connection and parse the query than to insert a row. It's important to cut redundancy by not opening and closing the connection unnecessarily.

Normalization Check

Our project requires multivalued-variables, but each cell in the table still only has 1 value. The 1st Normal Form requires each cell to only have 1 value. Additionally, each element is unique. Our database satisfies the 1st Normal Form. Our database contains attributes which at first seem like they have dependency on other attributes.

However, these variables are either calculated or given initial values and updated later based on specific events. There are no relations that are not defined via foreign key and primary key relationships. This means that our attributes have no functional or transitive dependency on another non-prime attribute, so our database satisfies the 2nd and 3rd Normal Forms.

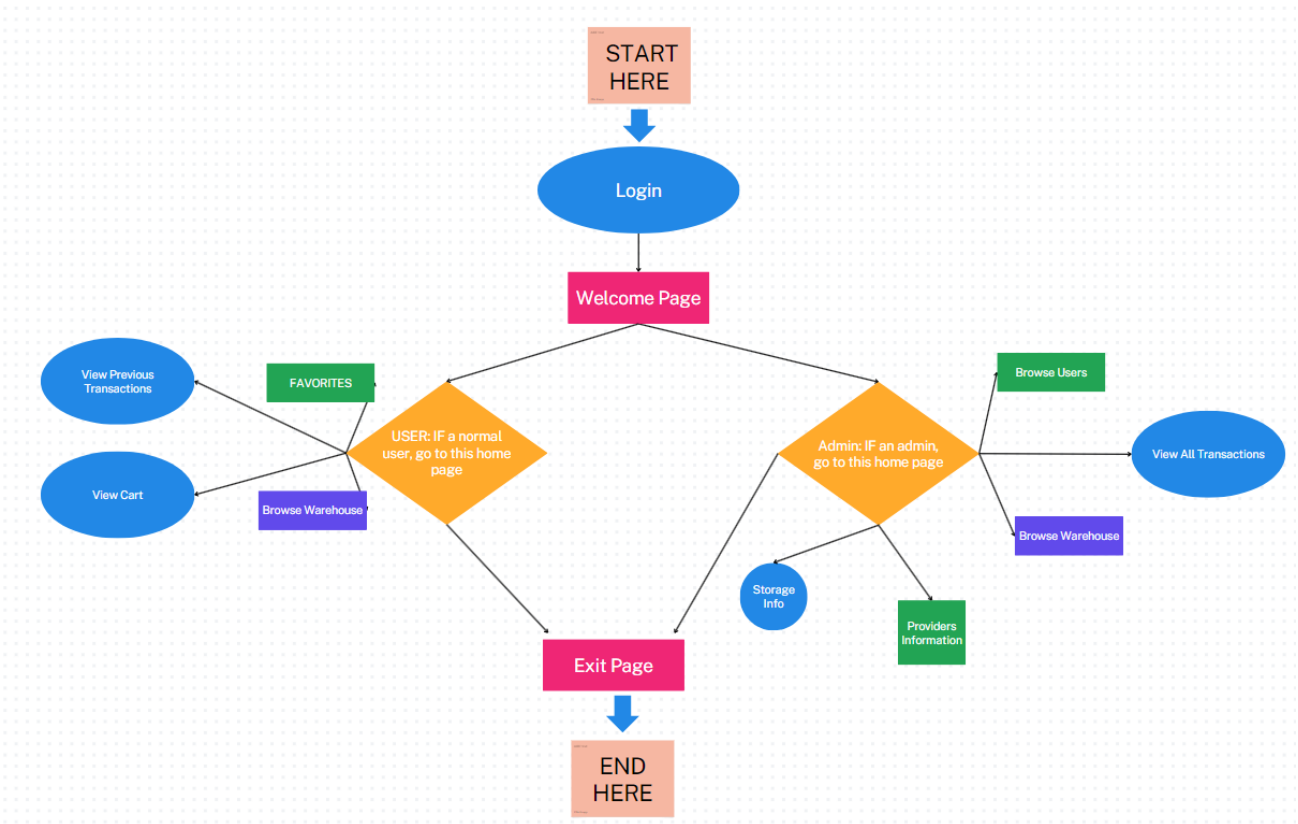
IX. Application Development

GUI Design

Figure 5 (GUI Diagram):

-Pages

- Login - allows users to login or create a new account. There will be an option to login or create an account as an admin. There will be a special universal password for admin accounts.
- Main Menu - contains a search bar and item list. Contains buttons to navigate to other pages.
- My Cart - displays the cost of the items that you are looking to buy. This page allows you to purchase items.
- My Account - view and edit information about your account.
- Transaction History - View your transaction history. Admins may view anyone's transaction history. Admins may sort by person or by item.
- Admin Log and Requests - admins may view important admin activity. Admins may accept or deny borrow requests.
- Update Warehouse - admins will have to manually add/update/remove items from the warehouse (except for items that are bought by customers). This is like the manual analog to scanning a barcode to add an item to the warehouse.



Views Implementation

-Description

-Updatability: All of our views are fully updatable. This is because when we go to the application of our code, if we update a table in our database we can still use the view and access the updated data. Since we need to add/edit/remove certain entries such as accounts, items in the warehouse, and others, we need our views to be updatable.

-Code

```
use warehouse;

create view vAccounts as select username, password, isAdmin, firstName, lastName,

address, email from accounts;
```

```
create view vAdminLog as select logID, adminUsername, description, timeStamp
from adminLog;
create view vCarts as select username, itemID, checkOutTIme, quantity, borrowing,
checkedOut from carts;
create view vFavorites as select favoriteID, username, itemID from favorites;
create view vItems as select itemID, type, name, providerID, stockQuantity, placeID,
pricePerUnit from items;
create view vProviders as select providerID, name, email from providers;
create view vStoragePlaces as select placeID, aisle, shelf from storagePlaces;
create view vTransactionLog as select transactionID, username, checkoutTime,
expectedDeliveryTime, actualDeliveryTime, borrowReturnTime, borrowState,
totalCost, creditCardNum from transactionLog;
```

Graphical User Interface

I. Connection to Database

A. Description

1. The code consists of a Node.js application using Express to handle account-related operations. The MySQL database connection is managed in mysql.js. accountController.js defines functions for checking login credentials, creating accounts, retrieving current accounts, and deleting records. It utilizes the MySQL database for querying and updating account information. The code uses asynchronous functions and promises to interact with the database.

The MySQL connection details are specified in mysql.js. Additionally, there's a basic error handling mechanism for the database connection.

B. Code:

accountController.js:

```
//const account = require('../Modules/account');
const mysql = require('../mysql');
const adminPassword = 'admin123';
async function checkLogin(req, res) {
    let uName = req.body.username;
    let pass = req.body.password;
    let admin = req.body.isAdmin;
    let adminPass = req.body.adminPassword;
    const allAccounts = await mysql.selectQuery("select username, password, isAdmin from accounts");
    let response = {
        loginValid: false,
        message: "Username not found"
    };
    for (let acct of allAccounts) {
        if (acct.username === uName) {
            response.message = "Incorrect password";
            if (acct.password === pass) {
                if (acct.isAdmin && admin) {
```

```
                response.message = "Incorrect admin password";
                if (adminPass === adminPassword) {
                    response.loginValid = true;
                    response.message = "Login successful";
                }
            } else {
                response.loginValid = true;
                response.message = "Login successful";
            }
        }
        break;
    }
  }
  res.send(response);
}
async function checkCreateAccount(req, res) {
  let uName = req.body.username;
  let pass = req.body.password;
  let admin = req.body.isAdmin;
  let adminPass = req.body.adminPassword;
  const allAccounts = await mysql.selectQuery("select username, password, isAdmin from
accounts");
  let response = {
```

```
        createValid: false,
        message: ""
    };
    for (let acct of allAccounts) {
        if (acct.username === uName) {
            response.message = "Username already exists";
            res.send(response);
            return;
        }
    }
    if (admin && adminPass !== adminPassword) {
        response.message = "Incorrect admin password";
            res.send(response);
            return;
    }
    response.createValid = true;
    response.message = "Account created";
    mysql.insertQuery("insert into accounts(username, password, isAdmin) values ?", [[uName, pass,
admin]]);
    res.send(response);
}
function getCurrAccount(req, res) {
    res.send(account.getCurrAccount());
```

```
}
function deleteRecord(req, res) {
    try {
        res.status(204).send(); // Respond with 204 (No Content) on successful deletion
    } catch {
        res.status(404).send({error: 'Rrecord not found'});
    }
}
module.exports = {
    checkLogin,
    checkCreateAccount,
    getCurrAccount,
    deleteRecord
};
```

mysql.js:

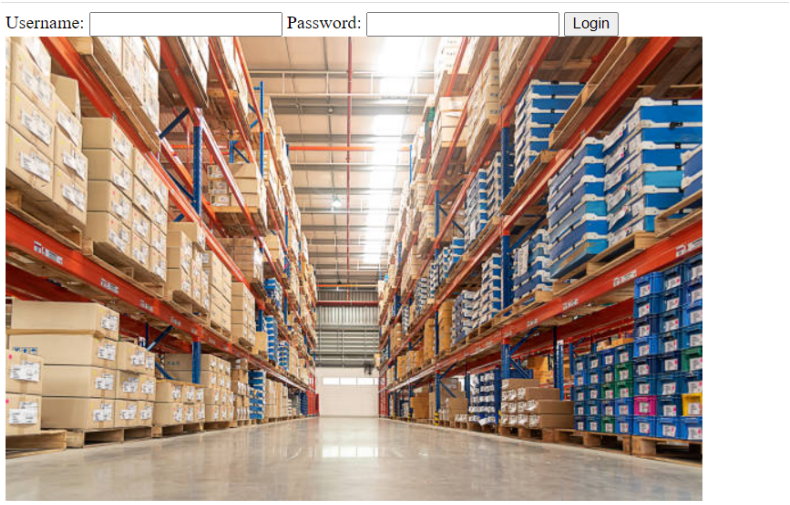
```
const mysql = require('mysql2');
const connection = mysql.createConnection({
    host: 'localhost',
            user: 'root',
            password: 'root',
            database: 'warehouse'
});
connection.connect((error) => {
```

```
    if(error){
      console.log('Error connecting to the MySQL Database');
      return;
    }
    console.log('Connection established sucessfully');
});
async function selectQuery(sql) {
    return new Promise((resolve, reject) => {
        connection.query(sql, function (err, result, fields) {
            if (err) reject(err);
            console.log("Selected from database");
            resolve(result);
        });
    });
}
function insertQuery(sql, values) {
    connection.query(sql, [values], function (err, result) {
        if (err) throw err;
        console.log("Number of records inserted: " + result.affectedRows);
    });
}
function updateQuery(sql) {
    connection.query(sql, function (err, result) {
```

```
        if (err) throw err;
        console.log("Number of records updated: " + result.affectedRows);
    });
}
function deleteQuery(sql) {
    connection.query(sql, function (err, result) {
        if (err) throw err;
        console.log("Number of records deleted: " + result.affectedRows);
    });
}
module.exports = {
    selectQuery,
    insertQuery,
    updateQuery,
    deleteQuery
};
```

II. Login Page

A. Here, users input their username and password. If correct, an alert box pops up with their login information and they are automatically redirected to the home menu. Otherwise, an alert box pops up “incorrect login attempt” and they are not able to log in. An image is shown of our logo, a warehouse.



B. Code:

1. HTML:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Login</title>
        <script src="./login.js"></script>
        <link rel="stylesheet" href="../css/index.css">
    </head>
    <body>
        <form id="loginForm">
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" required>

            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>

            <button type="button" onclick="login()">Login</button>
        </form>
        <img src="./warehousePhoto.jpg" alt="Warehouse here">
    </body>
</html>
```

2. CSS:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
```

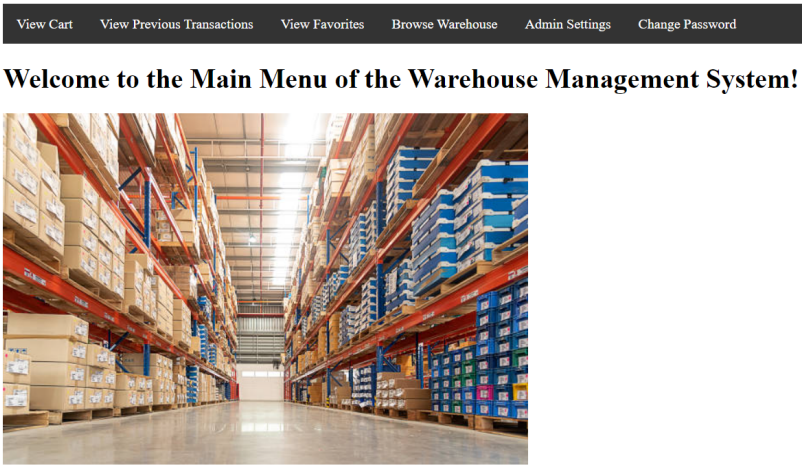
```
    margin: 0;
    display: flex;
    align-items: center;
    justify-content: center;
    height: 100vh;
}

form {
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    width: 300px;
}

label {
    display: block;
    margin-bottom: 8px;
}

input {
    width: 100%;
    padding: 8px;
    margin-bottom: 16px;
    box-sizing: border-box;
}

button {
    background-color: #4caf50;
    color: #fff;
    padding: 10px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

button:hover {
    background-color: #45a049;
}
```

3. JS

```
function login() {
    // You can perform any login logic here if needed
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;

    alert(`Login attempt\nUsername: ${username}\nPassword: ${password}`);
    // Redirect to another HTML page
    window.location.href = "home.html";
}
```

III. Main Menu Page

A. Provide buttons to redirect users to various pages including to Access Warehouse, Access Cart, View Previous Transactions, Access Favorites, Access Admin Settings (if admin), and Change Account Settings (password for users). There is an image of our logo, the warehouse.



B. Code:

1. HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="home.css">
    <title>Main Menu!</title>
</head>
<body>

    <ul>
```

```
        <li><a href="cart.html">View Cart</a></li>
        <li><a href="previousTrans.html">View Previous Transactions</a></li>
        <li><a href="favorites.html">View Favorites</a></li>
        <li><a href="warehouse.html">Browse Warehouse</a></li>
        <li><a href="admin.html">Admin Settings</a></li>
        <li><a href="changePass.html">Change Password</a></li>
    </ul>
    <h1> Welcome to the Main Menu of the Warehouse Management System!</h1>
    <img src="./warehousePhoto.jpg" alt="Warehouse here">
</body>
</html>
```

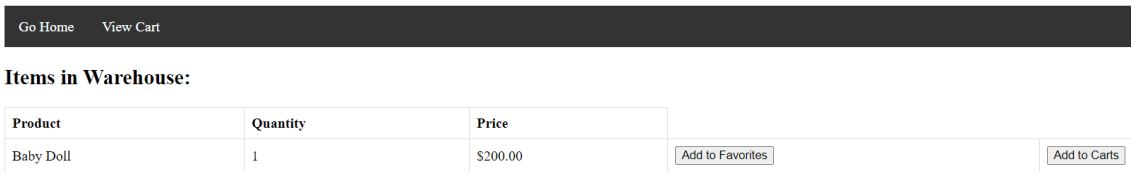
2. CSS:

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
  }

  li {
    float: left;
  }

  li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
  }

  /* Change the link color to #111 (black) on hover */
  li a:hover {
    background-color: #111;
  }
```

IV. Action Pages

A. View Warehouse

1. Here users can browse all of the items in the warehouse and view the items' product name, quantity, and price. They have the option to add these items to their favorites list or add the items to their cart. There is a menu where they can either return to the home menu or view their cart.



B. Code:

1. HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="warehouse.css">
    <title>Browse the Warehouse!</title>
</head>
<body>

    <ul>
        <li><a href="home.html">Go Home</a></li>
        <li><a href="cart.html">View Cart</a></li>
    </ul>
    <h2>Items in Warehouse:</h2>
    <!--
    <?php include("warehouse.php"); ?>
    -->

        <table>
        <thead>
            <tr>
                <th>Product</th>
                <th>Quantity</th>
                <th>Price</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Baby Doll</td>
```

```
                <td>1</td>
                <td>$200.00</td>
                <td><button>Add to Favorites</button></td>
                <td><button>Add to Carts</button></td>
            </tr>
        </tbody>
    </table>


</body>
</html>
```

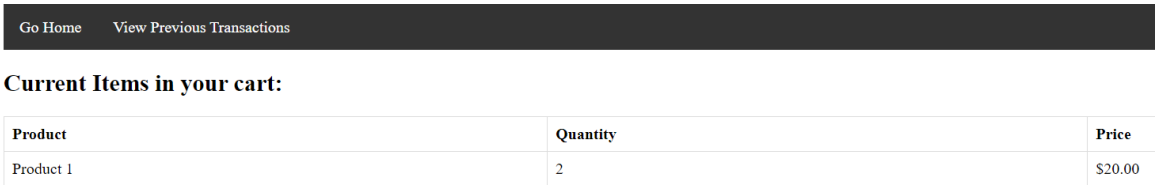
2. CSS:

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
  }

  li {
    float: left;
  }

  li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
  }

  /* Change the link color to #111 (black) on hover */
  li a:hover {
    background-color: #111;
  }

  /* Style the table */
table {
  border-collapse: collapse;
```

```
  width: 100%;
}

/* Style table headers and cells */
th, td {
  border: 1px solid #dddddd;
  text-align: left;
  padding: 8px;
}

/* Style alternating rows */
tr:nth-child(even) {
  background-color: #f2f2f2;
}
```

C. View Cart

1. Here, users can view their cart and see the items they intend to purchase (the product, quantity, and price). Here they can also redirect via a menu to either the home menu or the previous transactions page.



2. Code:
   a) HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="cart.css">
    <title>View your cart!</title>
</head>
<body>

    <ul>
        <li><a href="home.html">Go Home</a></li>
        <li><a href="previousTrans.html">View Previous Transactions</a></li>
    </ul>
```

```
    <h2>Current Items in your cart:</h2>
    <table>
        <thead>
            <tr>
                <th>Product</th>
                <th>Quantity</th>
                <th>Price</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Product 1</td>
                <td>2</td>
                <td>$20.00</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

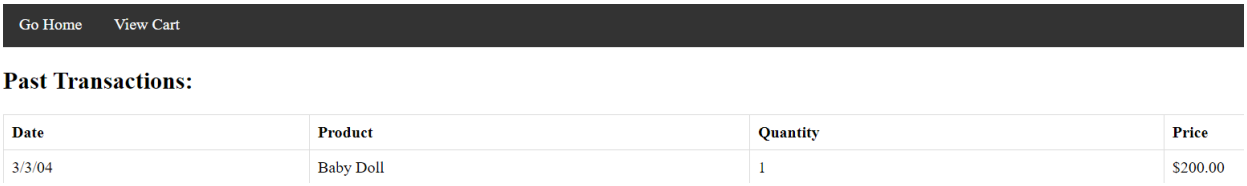
b) CSS:

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
  }

  li {
    float: left;
  }

  li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
  }
```

```
  /* Change the link color to #111 (black) on hover */
  li a:hover {
    background-color: #111;
  }

  /* Style the table */
table {
  border-collapse: collapse;
  width: 100%;
}

/* Style table headers and cells */
th, td {
  border: 1px solid #dddddd;
  text-align: left;
  padding: 8px;
}

/* Style alternating rows */
tr:nth-child(even) {
  background-color: #f2f2f2;
}
```

D. Previous Transactions

1. Here users can view a table of their previous transactions; including the date, product, quantity, and price. There is also a menu where they cdan redirect to the home menu or the cart.



2. Code:
    a) HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="previousTrans.css">
```

```
    <title>View your previous Transactions!</title>
</head>
<body>

    <ul>
        <li><a href="home.html">Go Home</a></li>
        <li><a href="cart.html">View Cart</a></li>
    </ul>
    <h2>Past Transactions:</h2>
    <table>
        <thead>
            <tr>
                <th>Date</th>
                <th>Product</th>
                <th>Quantity</th>
                <th>Price</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>3/3/04</td>
                <td>Baby Doll</td>
                <td>1</td>
                <td>$200.00</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

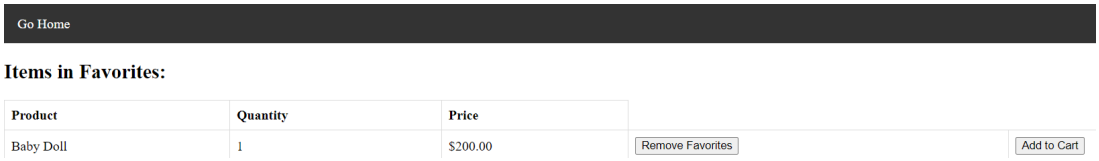
b) CSS:

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
  }

  li {
    float: left;
```

```
  }

  li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
  }

  /* Change the link color to #111 (black) on hover */
  li a:hover {
    background-color: #111;
  }

  /* Style the table */
table {
  border-collapse: collapse;
  width: 100%;
}

/* Style table headers and cells */
th, td {
  border: 1px solid #dddddd;
  text-align: left;
  padding: 8px;
}

/* Style alternating rows */
tr:nth-child(even) {
  background-color: #f2f2f2;
}
```

E. Favorites

1. This is a page listing the items that the user has labelled as their “favorites”. Here, the users have the ability to remove items from the favorites list and the ability to automatically add these items to their cart. There is also a menu to redirect to the home page.



2. Code:
    a) HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="favorites.css">
    <title>Browse Favorites!</title>
</head>
<body>

    <ul>
        <li><a href="home.html">Go Home</a></li>
    </ul>
    <h2>Items in Favorites:</h2>

        <table>
        <thead>
            <tr>
                <th>Product</th>
                <th>Quantity</th>
                <th>Price</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Baby Doll</td>
                <td>1</td>
                <td>$200.00</td>
                <td><button>Remove Favorites</button></td>
                <td><button>Add to Cart</button></td>
            </tr>
        </tbody>
    </table>
```

```
</body>
</html>
```

b) CSS:

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
  }

  li {
    float: left;
  }

  li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
  }

  /* Change the link color to #111 (black) on hover */
  li a:hover {
    background-color: #111;
  }

  /* Style the table */
table {
  border-collapse: collapse;
  width: 100%;
}

/* Style table headers and cells */
th, td {
  border: 1px solid #dddddd;
  text-align: left;
```

```
  padding: 8px;
}

/* Style alternating rows */
tr:nth-child(even) {
  background-color: #f2f2f2;
}
```

F. Admin Settings

a) This is only available to administrators in the WMS system. From this main page, admins are able to select whatever action they would like to perform. There are buttons to navigate to edit user data which allows admin to add and remove users. There's a button to view all transactions in a tabular format. There's an option to view all storage information and also all of the provider information. All of these are separate pages.

2. Change User Info
    a) View all current users and information in tabular form
    b) Ability to add, remove, and edit this information
    c) Option to redirect to Admin page, Home page
3. View All Transactions
    a) View all transactions in a tabular format
    b) Option to redirect to Admin page, Home page
4. Storage Info
    a) View all the information where items are stored in tabular format
    b) Option to redirect to Admin page, Home page
5. Provider Info
    a) View all the information on item providers in tabular format
    b) Option to redirect to Admin page, Home page
6. Code:
    a) HTML:

```
<!DOCTYPE html>
```

```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="admin.css">
    <title>Admin Menu</title>
</head>
<body>

    <ul>
        <li><a href="home.html">Go Home</a></li>
        <li><a href="#">Edit User Data</a></li>
        <li><a href="#">View All Transactions</a></li>
        <li><a href="#">View Storage Info</a></li>
        <li><a href="#">View Providers Info</a></li>

    </ul>
    <h1> Welcome to the Admin Menu of the Warehouse Management System!</h1>
    <h2> Choose what action you would like to perform </h2>
</body>
</html>
```

b) CSS:

```css
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
  }

  li {
    float: left;
  }

  li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
```

```
}

/* Change the link color to #111 (black) on hover */
li a:hover {
  background-color: #111;
}
```

G. Change Password

1. The change password page offers the ability for a user to change their password. After they input their username, current password, and their desired new password, they hit the button and an alert shares their new account information and they are automatically redirected to the home page

2. Code:
   a) HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="./changePass.js"></script>
    <link rel="stylesheet" href="changePass.css">
    <title>Admin Menu</title>
</head>
<body>

    <ul>
        <li><a href="home.html">Go Home</a></li>

    </ul>
    <h1> Change Password Page!</h1>
    <form id="loginForm">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required>
```

```
        <label for=" oldpassword">Current Password:</label>
        <input type="oldpassword" id="oldpassword" name="oldpassword"
required>

        <label for=" password">New  Password:</label>
        <input type="password" id="password" name="password" required>

        <button type="button" onclick="login()">Login</button>
    </form>
</body>
</html>
```

Sort Pages

- ## Outline
    - Here we will have the ability for our users to sort the items in the warehouse by column.
- ## Implementation
    - We will use a sort algorithm to ensure that users are able to sort here.
- ## Images

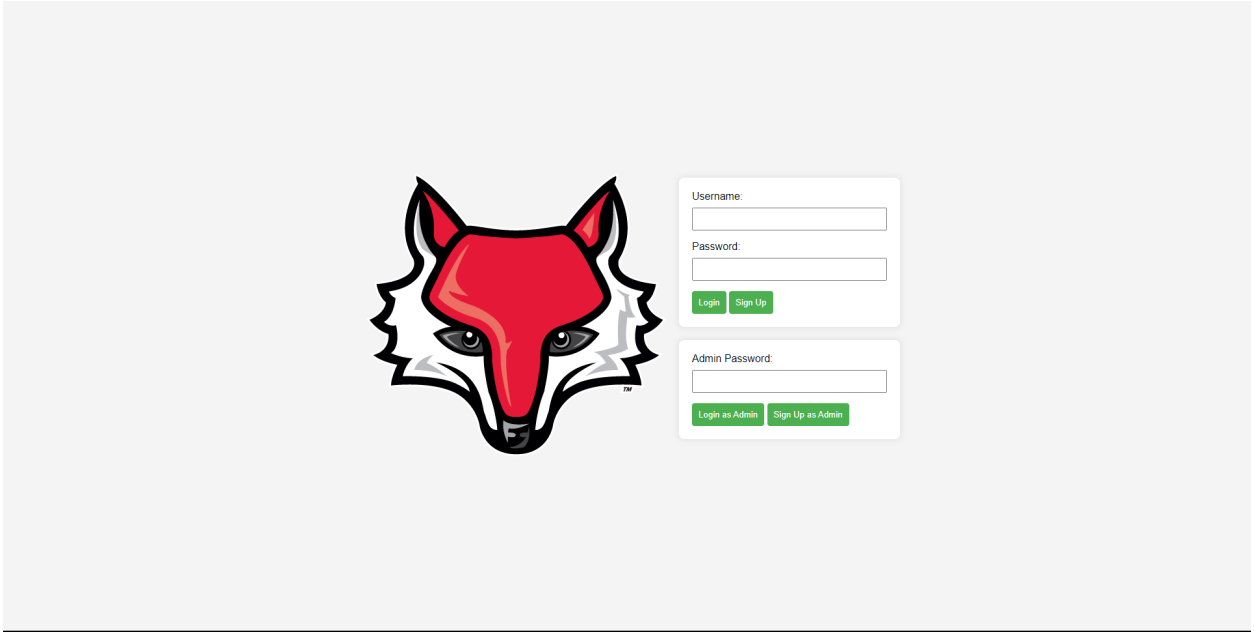
- ## Code

```
//It successfully stores whether the column was previously ascending or
descending
exports.sortCurrencies = function (req, res) {
    sortCategory = req.body.sortField;
    //sortCategory = "price";
    for (let x = previousCategory.length - 1; x >= 0; x--) {
        if (previousCategory[x] == sortCategory) {
            if (previousOrder[x] == true) {
                console.log("marked false");
                currentOrder = false;
                console.log(currentOrder);
            }
            else if (previousOrder[x] == false) {
                console.log("marked true");
                currentOrder = true;
                console.log(currentOrder);
            }
            break;
```

```
        }
    }
    console.log(req.body);
    if (currentOrder) {
        cryptoCurrencies.sort(function (a, b) { return (a[sortCategory] <
b[sortCategory]) ? 1 : -1 });
    }
    else {
        //is this correct to make it ascending?
        cryptoCurrencies.sort(function (a, b) { return (a[sortCategory] <
b[sortCategory]) ? -1 : 1 });
    }
    //originalData.data.sort();
    //console.log(cryptoCurrencies);
    res.send(JSON.stringify(cryptoCurrencies));
    previousCategory.push(sortCategory);
    previousOrder.push(currentOrder);
}
```

Insertion Pages

- ## Outline
    - There are several ways that users can insert data into the warehouse management system database via our GUI. To insert data into the warehouse, user, storage, and provider info tables the user needs to be an admin. This is accessible via the AdminWarehouse, AdminLog, AdminTransactionLog and index pages. Through this the administrator can manipulate data via our GUI.
- ## Images



- ## Implementation
    - We use servers and controllers through javascript files. How it works is we call the controllers through js files for each html page, and those controllers are called from the server.js whenever a user puts in /warehouse or a similar api call into their url. These controllers perform GET, POST,

DELETE, and similar calls onto our database. This allows for all manipulation of the data and ensures the websites are constantly updated.

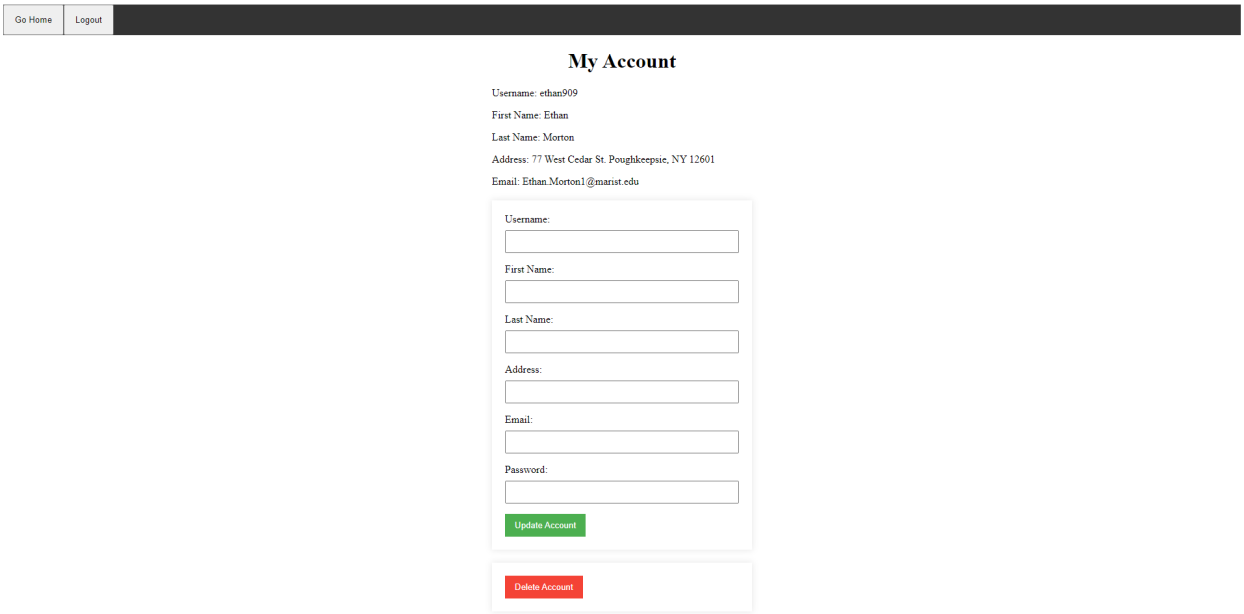
- **Code**

```
function addItem() {
  // Add fake item data
  var fakeItem = {
    id: 7,
    product: 'New Product',
    quantity: 5,
    price: '$100.00',
  };

  // Add the new item to the table
  var table = document.querySelector('table tbody');
  var newRow = table.insertRow(table.rows.length);
  newRow.innerHTML = `<td><button onclick="openModal(${fakeItem.id})">${fakeItem.id}</button></td>
                <td>${fakeItem.product}</td>
                <td>${fakeItem.quantity}</td>
                <td>${fakeItem.price}</td>
                <td>
                  <button onclick="editItem(${fakeItem.id})">Edit Item</button>
                  <button onclick="removeItem(this)">Remove Item</button>
```

```
            </td>`;
    }
```

Modification Pages

- ## Outline
    - To modify the data, which primarily occurs in the warehouse table of our database, the user must be an administrator. Here, admins can access the warehouse via the adminWarehouse page. In this page they are able to click the "Edit Data" button and update the information about particular items. This is then automatically update into the WMS database
- ## Images



- ## Implementation
    - We use servers and controllers through javascript files. How it works is we call the controllers through js files for each html page, and those controllers are called from the server.js whenever a user puts in /warehouse or a similar api call into their url. These controllers perform GET, POST, DELETE, and similar calls onto our database. This allows for all manipulation of the data and ensures the websites are constantly updated.

- **Code**

```
function openModal(id) {
    var modal = document.getElementById("myModal");
    var modalContent = document.getElementById("modalContent");
    // Set the content of the modal (you can customize this)
    modalContent.innerHTML = 'Content for ID ' + id;

    modal.style.display = "block";
}
/**@deprecated*/
function closeModal() {
    var modal = document.getElementById("myModal");
    modal.style.display = "none";
}
/**@deprecated*/
function editItem(id) {
    var table = document.querySelector('table tbody');
    var rows = table.getElementsByTagName('tr');
    for (var i = 0; i < rows.length; i++) {
        var rowId = rows[i].getElementsByTagName('button')[0].textContent;
        if (rowId === id.toString()) {
            makeRowEditable(rows[i]);
```

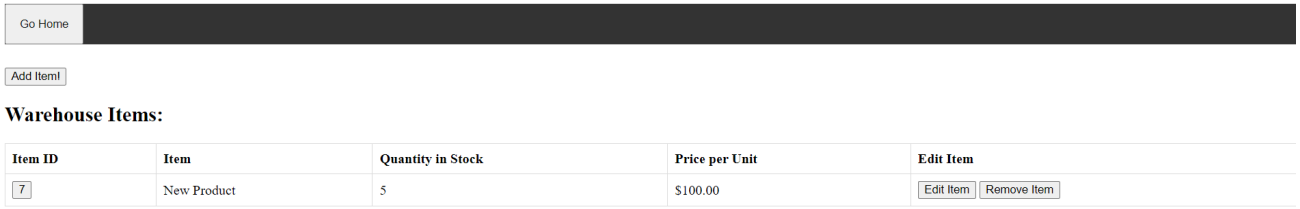
```
            break;
        }
    }
}
/**@deprecated*/
function makeRowEditable(row) {
    var cells = row.getElementsByTagName('td');
    var originalRowHTML = row.innerHTML;

    for (var i = 1; i < cells.length - 1; i++) {
        var content = cells[i].textContent;
        var input = document.createElement('input');
        input.type = 'text';
        input.value = content;
        cells[i].innerHTML = '';
        cells[i].appendChild(input);
    }
    var actionsCell = cells[cells.length - 1];
    actionsCell.innerHTML = `
<button onclick="saveChanges(this)">Save Changes</button>
<button onclick="cancelEdit(this, '${originalRowHTML}')">Cancel</button>`;
}
/**@deprecated*/
```

```
function saveChanges(button) {
    var row = button.parentNode.parentNode;
    var cells = row.getElementsByTagName('td');
    for (var i = 1; i < cells.length - 1; i++) {
        var input = cells[i].getElementsByTagName('input')[0];
        if (input) {
            cells[i].textContent = input.value;
        }
    }
    var actionsCell = cells[cells.length - 1];
    actionsCell.innerHTML = `
<button onclick="editItem(${row.cells[0].textContent})">Edit Item</button>
<button onclick="removeItem(this)">Remove Item</button>`;
}

/**@deprecated*/
function cancelEdit(button, originalRowHTML) {
    var row = button.parentNode.parentNode;
    row.innerHTML = originalRowHTML;
}
/**@deprecated*/
function removeItem(button) {
    var row = button.parentNode.parentNode;
```

```
    row.parentNode.removeChild(row);

}
```

Deletion Pages

- ## Outline
    - To delete data, which is primarily used in the “favorites” , “carts”, and “warehouse” pages, administrators are able to remove items from the warehouse page which are reflected in the database. Additionally, users have the ability to remove items from their favorites and their carts. These are all reflected in the WMS database.
- ## Images



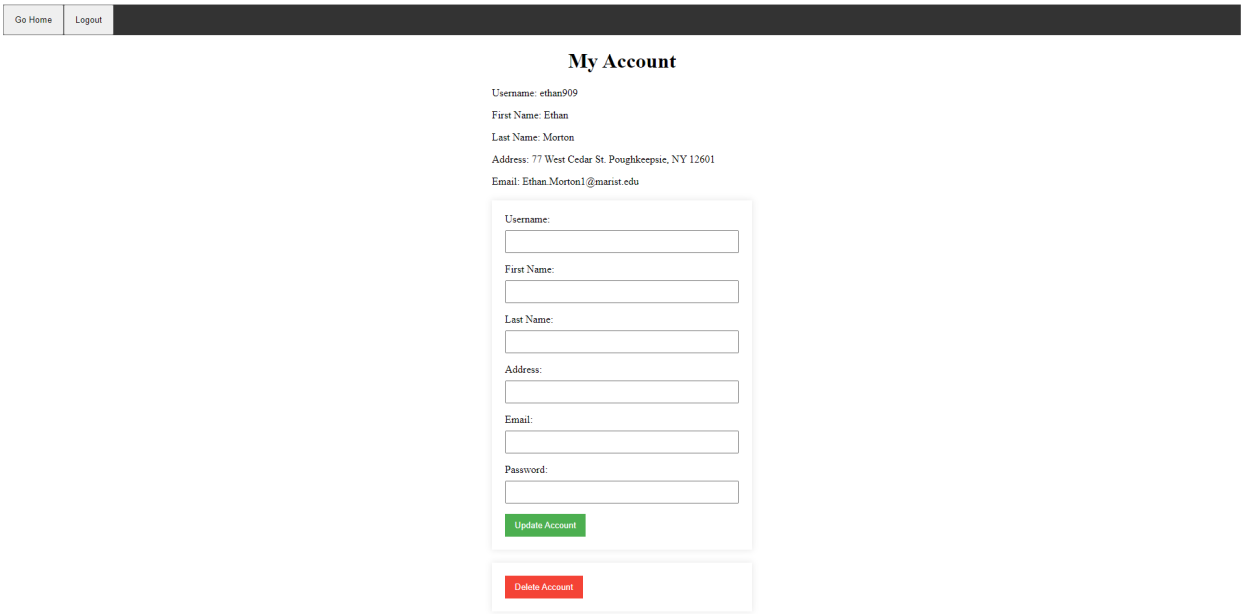
- ## Implementation
    - We use servers and controllers through javascript files. How it works is we call the controllers through js files for each html page, and those controllers are called from the server.js whenever a user puts in /warehouse or a similar api call into their url. These controllers perform GET, POST, DELETE, and similar calls onto our database. We use the DELETE function to delete items from the database.
- ## Code

```
function removeItem(button) {
  // Get the table row containing the clicked button
```

```
    var row = button.parentNode.parentNode;


    // Remove the row from the table
    row.parentNode.removeChild(row);
}
```

Print All Data

- ## Outline
    - To print out the data what we do is that when each page is loaded we ensure that the pages are automatically updated with information from the database to make sure that it is always up to date. This is our version of “printing out ” the information because it is always updated in the tables of the webpages.
- ## Images



- ## Implementation
    - We use servers and controllers through javascript files. How it works is we call the controllers through js files for each html page, and those controllers are called from the server.js whenever a user puts in /warehouse or a similar api call into their url. These controllers perform GET, POST, DELETE, and similar calls onto our database. We use the GET calls to retrieve data to display on

our various pages. We ensure that we call document.addEventListener('DOMContentLoaded'') to make sure that it is always updated when the page is loaded.

- **Code**

```
document.addEventListener('DOMContentLoaded', () => {
  RefreshTable();
});
async function RefreshTable() {
  try {
    const response = await fetch('/api/warehouse', {
      method: 'GET',
      headers: { 'Content-Type': 'application/json' }
    });
    if (!response.ok) {throw new Error('Network response was not ok');}
    const data = await response.json();
    console.log(data);
    if (data.success) {
      // Assume you have an array containing warehouse data called "Items"
      const warehouse = data.Items;
      // Get the table body element
      const tableBody = document.querySelector('tbody');
      // Populate the table with item data
      warehouse.forEach(item => {
```

```
            const row = document.createElement('tr');
            row.id = `itemEntry${item.itemID}`;
            //const descriptionCell = document.createElement('td');
            row.appendChild(document.createElement('td')).textContent = item.itemID;
            row.appendChild(document.createElement('td')).textContent = item.type;
            row.appendChild(document.createElement('td')).textContent = item.name;
            row.appendChild(document.createElement('td')).textContent = item.providerID;
            row.appendChild(document.createElement('td')).textContent = item.placeID;
            row.appendChild(document.createElement('td')).textContent = item.pricePerUnit;

            /**
             * Don't forget to create the button to edit this item
             */
            tableBody.appendChild(row);
        });
    }
  } catch (error) {
    console.error('Error fetching warehouse data:', error);
    alert('An error occurred while fetching warehouse data');
  }
}
```

X. References

1. ***BR Williams Warehouse Management System*** ***link*** — page 8

2. ***Koha Management System*** ***link*** — page 8

3. ***ShipHero Warehouse Management System*** ***link*** — page 8