

Project Report: Warewise

Natalie Kao (015551354, natalie.kao@sjsu.edu), Lac-Phong Nguyen (015625155, lac-phong.nguyen@sjsu.edu), Rohith Iyengar (015169232, rohith.iyengar@sjsu.edu), Shreya Raj(015466802, shreya.raj@sjsu.edu), Nyun Ei Hlaing (016460886, nyunei.hlaing@sjsu.edu)

Department of Computer Science, San José State University

CS 157A: Intro to Database Management Systems

Prof. Ramin Moazeni

May 10, 2024

Goals and Descriptions

Small businesses frequently encounter challenges in inventory management, primarily due to laborious manual processes like spreadsheet usage, ineffective tracking methods resulting in stockouts, and complex order management causing delays in transactions with customers and suppliers. Establishing a well-structured and efficient inventory system is crucial for the smooth operation and success of these enterprises. Warewise provides a user-friendly and automated inventory management solution as the application streamlines the tracking of products, suppliers and orders, making it easier for businesses to manage their inventory levels, process sales, and purchase orders. The application stores relevant information including employee information, overall balance, order details, and supplier information to facilitate informed decision making and seamless business operations. By automating these critical operations, Warewise's goal is to help small businesses enhance efficiency, reduce costs, and improve customer satisfaction.

Application/Functional Requirements/Architecture

Application:

Our project, Warewise, delivers a user-friendly and automated inventory management solution tailored for businesses. By centralizing all product-related operations onto one intuitive platform, Warewise streamlines the tracking of products, suppliers, and orders. It simplifies inventory management tasks, such as monitoring stock levels, processing sales and purchase orders, and generating insightful reports. With its inventory page displaying comprehensive product details including category, stock level, price, and reorder quantities, Warewise ensures businesses stay informed about their inventory status at all times. Its seamless integration with suppliers enables ordering directly from the platform, with newly ordered products automatically reflected in the inventory. Additionally, Warewise introduces a customer transactions page to facilitate sales and maintain customer records, along with features like an employee directory and an account page for managing business information. Overall, Warewise serves as a novel tool designed to enhance organization and efficiency for businesses, offering a centralized solution to meet their diverse needs.

Functionality:

Product Catalog/ Inventory:

Warewise boasts a comprehensive product catalog that encompasses essential details such as product items, categories, prices, and real-time stock levels, as well as reorder levels to anticipate demand and prevent under or overstock situations. This inventory ensures easy access to vital supply information, allowing businesses to visualize their products in a user friendly format.

Suppliers and Placing Orders:

Warewise facilitates the seamless processing of additional suppliers and orders by providing an interface for creating and managing them. By placing their orders through the application, businesses can have new products directly updated in their inventory and available for sale. This functionality also expands businesses' sourcing options by allowing them to add multiple suppliers to choose from, and fosters simpler communication and interaction.

Customer Transactions:

Warewise includes a dedicated transactions feature for managing customer transactions, allowing businesses to efficiently process sales, track customer interactions, and maintain comprehensive records. This feature not only facilitates smoother sales processes but also enables businesses to better understand their customers' preferences and behaviors. Every transaction is automatically reflected in the inventory, increasing efficiency as well.

Additional functionality:

In addition to its core functionality (mentioned above), Warewise provides some other features that make it stand out. The employee page is an example of this, and allows businesses to easily access employee records and information. Additionally, users of Warewise must register their accounts and log into the application when using it. This security measure introduces more confidentiality, and implements user authentication to ensure that each unique business has its own information correctly fetched and displayed (using a generated business ID). The business's personal information is then available on the account page for personalization and reference purposes.

Overall, Warewise's functionality encompasses the core aspects of inventory management and sales, offering businesses a robust platform for their growing operations.

Architecture of frontend:

As for the architecture of our application the pages necessary are as follows:

- Login
- Register
- Account
- Add Supplier
- Inventory
- Order
- Employee
- Transaction

The application consists of three main components: client, server, and database.

- The client side handles the user interface.
- The server side manages business logic and data processing.
- The database stores all relevant data.
- The application is primarily developed using JavaScript.

Database Design:

- The system uses 10 overarching tables to manage user information securely.
- By employing foreign keys linked to the business ID, the architecture ensures that each business accesses only its relevant data, enhancing data privacy and integrity.

Purpose-Driven Architecture:

- This approach facilitates efficient data management and retrieval by minimizing data overlap and enhancing query performance.
- It also simplifies maintenance and scalability by keeping the database structure straightforward and aligned with business needs.

Security and Isolation:

- The use of business-specific IDs as foreign keys not only tailors the user experience but also adds a layer of security by isolating data access within the business scope.
- This design choice is crucial for protecting sensitive business information and preventing unauthorized data breaches.

The Login and Registration is an essential part of our application as all the backend functionality is dependent on the user first existing. Without a user registered, we are not able to fetch the suppliers for the user which we need to make orders, which we need to get products, which we need to make transactions with customers. The registration page will be where the user starts and after registering with us they will be able to login.

The Login/Registration would set up a row for that business in our overarching business table with the username and a hashed password along with other attributes stored in the table. This is done using a POST request on the database, which makes a query to add a new row to the business table. A business id is automatically generated and used to ensure that all the foreign key relations from other tables are properly connected to the business.

From the login they are sent to a general account page where they will be able to see all their account information such as their username, business name, address, balance, and the date in which the account was created.

The information on the account page is acquired through creating a GET request, which sends makes a query into our database and grabs information from the business as well as the balance tables. Then, these tables are joined on business id and sent back to the caller as JSON body, which is then displayed on the page.

Though the user can visit the inventory and order pages at this time, these pages will remain empty as no order can be made without a supplier and no products can be added into the inventory until an order is made. So an add supplier page is necessary. On this page the supplier will need to give a supplier name, the category of supplier, a valid phone number, email, and address.

The inventory page works by creating a GET request which makes a query on the database grabbing data from the products table only relevant to that business. We make sure that only the data for that business is received by making sure the business id is sent and included in the query. The product data is received and then displayed on the page. As for suppliers, we make a similar POST request to login/registration to add a new supplier for the business. For this too, we send the business id through the request.

Now we are able to go to the order page. Here a list of existing orders will be displayed for the user to see. The user is also able to make a new order where they will be able to select a supplier and input a product that they want from the supplier. The user will have to provide the supplier the product's name, category, description, as well as the quantity and price of the product. After doing so, the user can click on the button to make the order and the order will be registered and displayed to the user.

On the backend, orders works similarly to suppliers by making a POST request which uses an insert statement to add a new row and update the business orders suppliers and business product suppliers junction tables.

Now that the user made an order, the product that they requested from the supplier can now be displayed for the user to see. The product information is stored under a table that shows the user the product name, ID, description, quantity of the product, and the price of the product. The products are all under their respective categories and sorted as such. The user is also able to select which category of products they want to display on the page.

With products in the business's inventory, customers are now able to make transactions. To make a transaction valid, the user will have to input the product's ID, the quantity of the product in the transaction, and the first and last name of the customer as well as their email, phone, address, and payment method. Once submitted, the transaction will be recorded and the product's quantity will be updated in the inventory tab of our application.

Finally there is the employee page, where users will be able to manage their employees. Users are able to add employees (in which their first name, last name, email, phone number, address, and salary will need to be inputted) and delete them as they see fit. The page will contain all employees under the business and display all their information and ID number.

If our application were to be deployed for an actual business, some of these pages will have to be changed to fit a more realistic situation. Adding a supplier would not be relevant in actual practice, as our application instead will have two different logins, one for a business and one for a supplier. This way, our application for suppliers would provide them a way to add products that they are selling themselves and for businesses they would be able to view all the suppliers and the products that these suppliers offer and order like that instead of manually

inputting in products from a supplier. But given time constraints, scope, and scale of our project, we decided to stick with this design on the frontend side of things as this design allows us to easily test the backend features of our application.

As for the backend side, we can improve our table designs to be better suited to handle more functions for the business, including the things mentioned above, which may require changes on the backend. However, considering the scope of the project, we were limited to this functionality.

The diagram illustrates a business system with the following entities and their attributes:

- Balance**: balance_id, balance, timestamp, business_id
- Business**: business_id, business_name, username, password, creator_date, address, creator
- Customers**: customer_id, first_name, last_name, email, address, phone, business_id
- Employees**: employee_id, first_name, last_name, salary, address, phone, email, business_id
- Orders**: order_id, order_line_id, order_date, price, quantity, product_name, business_id
- Products**: product_id, category_name, product_name, quantity, price, reorder_level, reorder_quantity, product_description, business_id
- Sales**: sales_id, business_id, product_id, quantity, price, order_date, payment_details
- Suppliers**: supplier_id, supplier_name, phone, email, address, supplier_category, business_id
- Business_Product_Supplier**: business_id, product_id, supplier_id
- Business_Orders_Suppliers**: business_id, order_id, supplier_id

Relationships (diamonds) and their cardinalities:

- creates**: Business (1) to Balance (1)
- has**: Business (1) to Orders (1)
- has**: Business (1) to Products (1)
- has**: Business (1) to Sales (1)
- has**: Business (1) to Business_Product_Supplier (1)
- has**: Business (1) to Business_Orders_Suppliers (1)
- manages**: Business (1) to Employees (1)
- generates**: Business (1) to Sales (1)
- provided by**: Business (1) to Suppliers (1)
- has**: Business_Product_Supplier (1) to Suppliers (1)
- has**: Business_Orders_Suppliers (1) to Suppliers (1)
- has**: Business_Orders_Suppliers (1) to Orders (1)
- has**: Business_Product_Supplier (1) to Products (1)

Original Tables:

1. BUSINESS (BUSINESS_ID: Integer, USERNAME: String, PASSWORD: String, BUSINESS_NAME: String, ADDRESS: String, CREATION_DATE: Timestamp)
2. EMPLOYEES (EMPLOYEE_ID: Integer, BUSINESS_ID: Integer, FIRST_NAME: String, LAST_NAME: String, EMAIL: String, PHONE: String, ADDRESS: String, SALARY: Decimal)
 - BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS

3. PRODUCTS_SUPPLIERS_ORDERS (BUSINESS_ID: Integer, PRODUCT_ID: Integer, SUPPLIER_ID: Integer, ORDER_ID: Integer, PRODUCT_NAME: String, PRODUCT_DESCRIPTION: String, PRODUCT_QUANTITY: Integer, REORDER_LEVEL: Integer, REORDER_QUANTITY: Integer, PRODUCT_PRICE: Decimal, SUPPLIER_NAME: String, EMAIL: String, PHONE: String, ADDRESS: String, SUPPLIER_CATEGORY: String, ORDER_QUANTITY: Integer, ORDER_PRICE: Decimal, ORDER_DATE: Timestamp)
 - BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
4. CUSTOMERS (CUSTOMER_ID: Integer, BUSINESS_ID: Integer, FIRST_NAME: String, LAST_NAME: String, EMAIL: String, PHONE: String, ADDRESS: String)
 - BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
5. SALES (SALES_ID: Integer, BUSINESS_ID: Integer, PRODUCT_ID: Integer, QUANTITY: Integer, PAYMENT_DETAILS: String, PRICE: Decimal, ORDER_DATE: Timestamp)
 - BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
 - PRODUCT_ID is a foreign key connecting to PRODUCT_ID of PRODUCTS
6. BUSINESS_ORDERS_SUPPLIERS (BUSINESS_ID: Integer, ORDER_ID: Integer, SUPPLIER_ID: Integer)
 - BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
 - ORDER_ID is a foreign key connecting to ORDER_ID of ORDERS
 - SUPPLIER_ID is a foreign key connecting to SUPPLIER_ID of SUPPLIERS
7. BALANCE (BALANCE_ID: Integer, BUSINESS_ID: Integer, BALANCE: Integer, TIMESTAMP: Timestamp)

- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS

Issues with original table:

Functional Dependencies

- Product_ID → Business_ID, Category_Name, Product_Name, Product_Description, Reorder_Level, Reorder_Quantity, Price
- Supplier_ID → Business_ID, Supplier_Name, Email, Phone, Address, Supplier_Category
- Order_ID → Business_ID, Order_Quantity, Order_Price, Order_Date

Normalization Analysis

- 1NF
 - Condition: All attributes have atomic (indivisible) values, and there are no repeating groups.
 - Analysis: The table is in 1NF as each column holds atomic values and rows are uniquely identifiable.
- 2NF
 - Condition: It is in 1NF and all non-prime attributes are fully dependent on the whole of a candidate key.
 - Analysis: The table is not in 2NF:
 - Category_Name, Product_Name, Product_Description, Product_Quantity, Reorder_Level, Reorder_Quantity, Product_Price are dependent only on Product_ID.

- Supplier_Name, Email, Phone, Address, Supplier_Category are dependent only on Supplier_ID.
 - Order_Quantity, Order_Price, Order_Date are dependent only on Order_ID.
- 3NF
 - Condition: It is in 2NF and no transitive dependencies exist.
 - Analysis: Transitive dependencies would be evaluated if it were in 2NF. However, attributes like Product_Description and Supplier_Category depend on Product_ID and Supplier_ID, respectively, and not on any superkey.
- BCNF (Boyce-Codd Normal Form)
 - Condition: It is in 3NF, and every determinant is a superkey.
 - Analysis: The table is not in BCNF because:
 - Product_ID and Supplier_ID are not superkeys but determine multiple attributes.

Conversion to BCNF:

To convert the original combined table to BCNF, follow these steps:

1. Identify and Separate Partial Dependencies: Attributes dependent solely on Product_ID are separated into a Products table. Similarly, attributes dependent solely on Supplier_ID are moved to a Suppliers table. Similarly, attributes dependent solely on Order_ID are moved to a Suppliers table. This step eliminates partial dependencies, addressing the issues of 2NF.

2. Resolve any Transitive Dependencies: Ensure that each attribute in the Products, Suppliers, and Orders tables depends only on the primary key of that table, not on other non-prime attributes. Additional decomposition is necessary here.
3. Ensure All Determinants are Superkeys: In the remaining tables, ensure that for every non-trivial functional dependency, the determinant is a superkey. This is achieved by defining appropriate primary keys for Products, Suppliers, and Orders, and by ensuring the composite key in BusinessProductSupplierOrders adequately controls any dependent attributes.

Changes for our tables:

- Create a Products Table
 - Ensures that product-related attributes are only dependent on the Product_ID.
 - Attributes: Product_ID, Business_ID, Category_Name, Product_Name, Product_Description, Quantity, Reorder_Level, Reorder_Quantity, Price
 - Primary Key: Product_ID
 - Foreign Key: Business_ID
- Create a Suppliers Table
 - Each supplier attribute solely depends on the supplier ID.
 - Attributes: Supplier_ID, Business_ID, Supplier_Name, Email, Phone, Address, Supplier_Category
 - Primary Key: Supplier_ID
 - Foreign Key: Business_ID
- Create an Orders Table
 - Manages order-specific details like quantity and price for every order.

- Attributes: Order_ID, Business_ID, Quantity, Price, Order_Date
- Primary Key: Order_ID
- Foreign Key: Business_ID
- Create a Business_Product_Suppliers Table
 - Handles relationships between Business, Products, and Suppliers.
 - Attributes: Business_ID, Product_ID, Supplier_ID
 - Composite Key: Business_ID, Product_ID, Supplier_ID
 - Foreign Keys: Business_ID, Product_ID, Supplier_ID
- Create a Business_Order_Suppliers Table
 - Handles relationships between Business, Orders, and Suppliers.
 - Attributes: Business_ID, Order_ID, Supplier_ID
 - Composite Key: Business_ID, Order_ID, Supplier_ID
 - Foreign Keys: Business_ID, Order_ID, Supplier_ID

Now, all attributes are dependent on the key, the whole key, and nothing but the key.

No partial or transitive dependencies are present.

Each determinant in functional dependencies is a candidate key or superkey.

Final Tables (All tables now in BCNF):

1. BUSINESS (BUSINESS_ID: Integer, USERNAME: String, PASSWORD: String, BUSINESS_NAME: String, ADDRESS: String, CREATION_DATE: Timestamp)
2. EMPLOYEES (EMPLOYEE_ID: Integer, BUSINESS_ID: Integer, FIRST_NAME: String, LAST_NAME: String, EMAIL: String, PHONE: String, ADDRESS: String, SALARY: Decimal)

- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
3. PRODUCTS (PRODUCT_ID: Integer, BUSINESS_ID: Integer, CATEGORY_NAME: String, PRODUCT_NAME: String, PRODUCT_DESCRIPTION: String, QUANTITY: Integer, REORDER_LEVEL: Integer, REORDER_QUANTITY: Integer, PRICE: Decimal)
- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
4. SUPPLIERS (SUPPLIER_ID: Integer, BUSINESS_ID: Integer, SUPPLIER_NAME: String, EMAIL: String, PHONE: String, ADDRESS: String, SUPPLIER_CATEGORY: String)
- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
5. BUSINESS_PRODUCT_SUPPLIER (BUSINESS_ID: Integer, PRODUCT_ID: Integer, SUPPLIER_ID: Integer)
- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
 - PRODUCT_ID is a foreign direct key connecting to PRODUCT_ID of PRODUCTS
 - SUPPLIER_ID is a foreign key connecting to SUPPLIER_ID of SUPPLIERS
6. CUSTOMERS (CUSTOMER_ID: Integer, BUSINESS_ID: Integer, FIRST_NAME: String, LAST_NAME: String, EMAIL: String, PHONE: String, ADDRESS: String)
- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
7. SALES (SALES_ID: Integer, BUSINESS_ID: Integer, PRODUCT_ID: Integer, QUANTITY: Integer, PAYMENT_DETAILS: String, PRICE: Decimal, ORDER_DATE: Timestamp)
- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS

- PRODUCT_ID is a foreign key connecting to PRODUCT_ID of PRODUCTS
8. ORDERS (ORDER_LINE_ID: Integer, ORDER_ID: Integer, BUSINESS_ID: Integer, PRODUCT_NAME: String, QUANTITY: Integer, PRICE: Decimal, ORDER_DATE: Timestamp)
- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
9. BUSINESS_ORDERS_SUPPLIERS (BUSINESS_ID: Integer, ORDER_ID: Integer, SUPPLIER_ID: Integer)
- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS
 - ORDER_ID is a foreign key connecting to ORDER_ID of ORDERS
 - SUPPLIER_ID is a foreign key connecting to SUPPLIER_ID of SUPPLIERS
10. BALANCE (BALANCE_ID: Integer, BUSINESS_ID: Integer, BALANCE: Integer, TIMESTAMP: Timestamp)
- BUSINESS_ID is a foreign key connecting to BUSINESS_ID of BUSINESS

Major Design Decisions

Modular Architecture:

By segregating client, server, and database components into distinct modules, Warewise adheres to the principle of separation of concerns, simplifying development and maintenance processes. Each component has its own responsibility and its own team members that worked to develop it, which increases readability and reusability. This modular approach not only facilitates easier updates but also lays the groundwork for potential scalability by segmenting the workload into manageable units. The frontend client component interacts with the database through API calls to retrieve and display information, and similarly updates the database's tables to store information. The majority of the calculations (such as current stock, prices for sales, etc) are calculated on the backend for ease, and the frontend retrieves the relevant information for each function of the application. It also creates a very user friendly approach, prioritizing efficient and adaptable features for the client. By implementing a more modular architecture, we were able to simplify both the development and usability of the application.

API Driven Communication:

Using an API-driven approach ensures that the system is highly extensible and can seamlessly integrate with other software tools, which is important for inventory systems that need to communicate with related platforms. By implementing an API driven architecture, Warewise efficiently exchanges data between the client and server, enabling features like real-time synchronization of inventory levels or automatic updating of financial records based on transaction data. This design choice not only enhances the flexibility and scalability of Warewise but also future-proofs the system to accommodate evolving business needs.

Choice of Javascript

By using Javascript for both the frontend and backend with Node.js, it enables a unified language across the stack, simplifying development and reducing the learning curve for new developers. This unified language allows seamless communication and collaboration between the frontend and backend team, as well as reducing the need for context switching between different programming languages. Javascript's versatility allows our team to build full-stack applications from the client-side interface to the server-side logic and simplifies the development process. By using Node.js as a server side Javascript runtime, it offers scalability, performance benefits, and handles connections efficiently.

Data Handling

Implementing a robust database management system is essential for maintaining data integrity and security. Using MySQL as our database system is crucial in handling transactions and queries efficiently, as it is fundamental for inventory management systems where real-time data processing is critical. By enforcing data integrity rules such as primary keys, foreign keys, auto increments, and unique constraints, MySQL helps maintain accuracy and consistency of data stored in the database. MySQL allows for efficient transactions for the application as its capability for executing multiple database operations as a single unit of work follows ACID properties, ensuring atomicity, consistency, isolation, and durability. This is crucial for Warewise, as accurate and reliable processing of transactions for sales, purchases, and inventory updates are essential. MySQL's scalability, high availability, compatibility, and ease of integration with Javascript allows businesses to ensure reliability and an effective management of inventory data.

Implementation Details

For this project we utilized a tech stack consisting of React native, MySQL, and Node.js/Express. React was used for implementing all the frontend pages while MySQL and Node.js/Express was used for implementing all the backend functionality of our application. The general schema of our backend was created through MySQL. The server for our application was developed with Node.js using the Express framework to seamlessly integrate frontend and backend communication. Our server consisted of mainly two components: API calls and SQL Query functions. In the database.js file are all the SQL queries needed for our application to work properly. For example, to register an account we needed to write an insert statement function so that a new business can be inserted into the Business table so that when the user hits the register button a new business is created with a business ID which is used throughout our application to ensure that all the operations done by the user (adding supplier, making orders, adding employees, etc.) is attached only specifically for that user. This way user's data are kept separate from each other and there's no confusing overlap between two different businesses. But these functions hold no purpose unless they are somehow called by the frontend, which is why we export these functions.

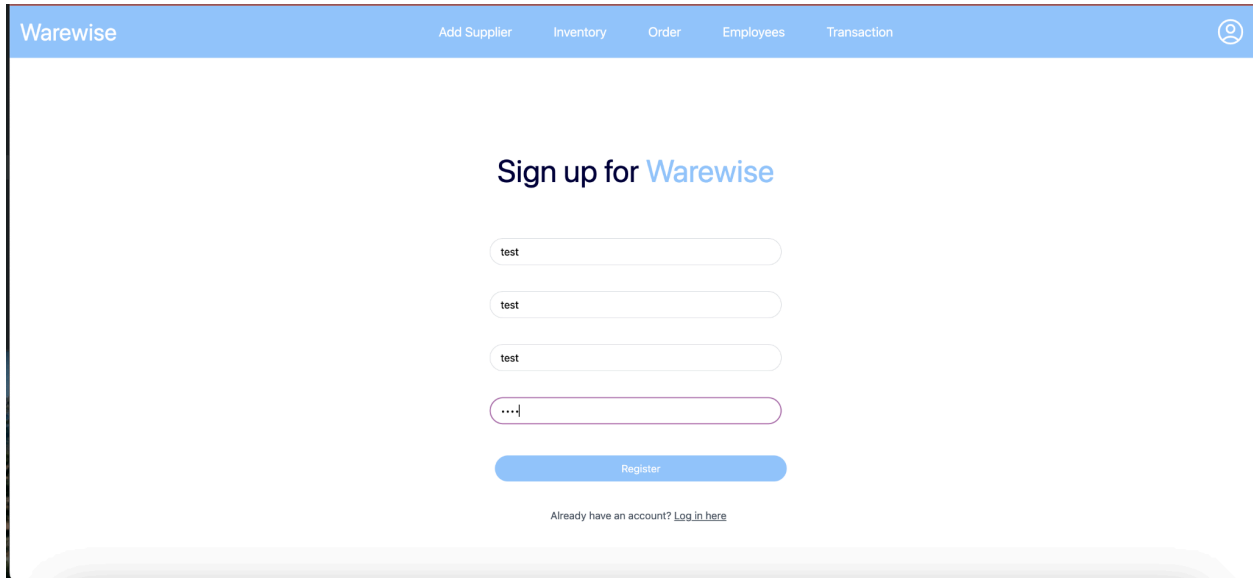
This is where the app.js file comes in. Here we create an app object that explicitly calls the Express framework, initializing the object as an Express application. In doing so, we are able to implement RESTful API calls throughout our application, handling and routing requests to and from the local database so that the frontend is able to access the information and send back information to the database. app.js also imports all the exported functions from database.js in order to use them to get/send info to the database. For example, in app.js we write a post request from app (the object initialized as an Express application) in which we route the request to

localhost:8080/login. Inside of this request we also create an asynchronous function which makes all SQL query function calls related to logging in from database.js and sends it to the local database. So now that we have a port on our local device that is handling all the requests to and from the database, all we need to do now is access that port from the frontend and our application will be fully functional.

To connect the frontend to the backend, we use Axios, a React library used to handle HTTP requests. With Axios, we are able to send get, post, delete, and other requests to the server and fetch/send all the data and information we need from the database. For example, on the Register page, we make a post request with Axios to the route localhost:8080/register and pass in the username, password, business name, and address that the user inputs on the frontend if they submit the register form. The server will then run the code associated with that route and call the SQL query function (in this case the insertBusiness) to the database and insert the new business into the Business table on the local database.

Demo

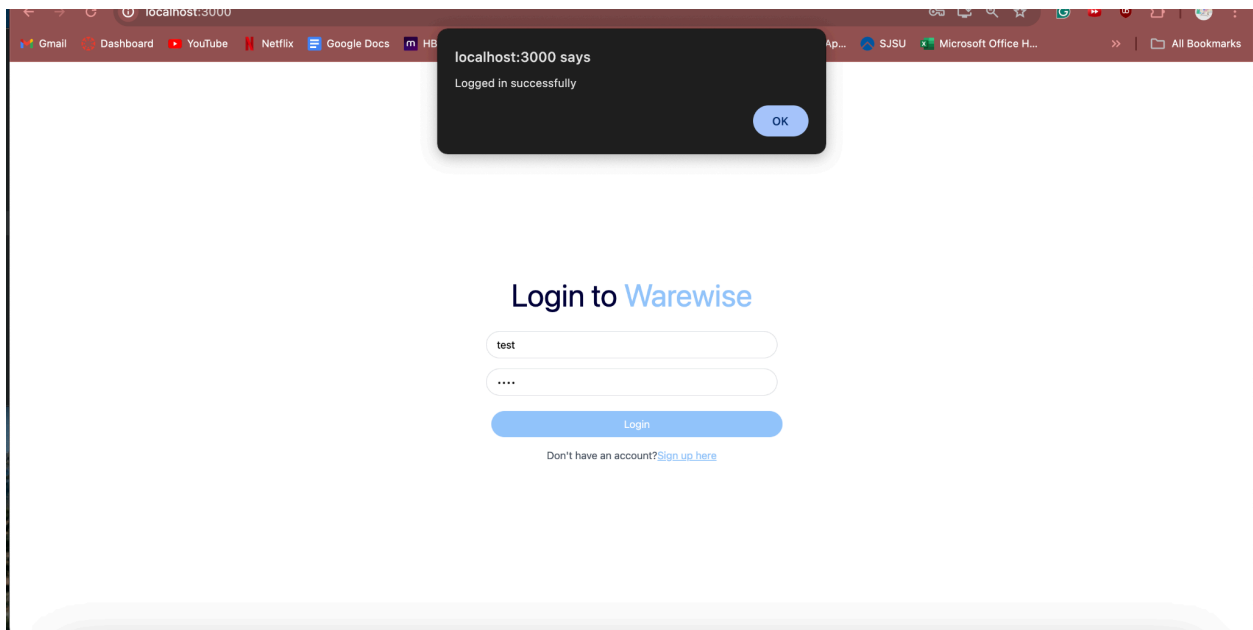
Register Page



The screenshot shows the 'Warewise' registration page. At the top is a blue navigation bar with the 'Warewise' logo on the left and links for 'Add Supplier', 'Inventory', 'Order', 'Employees', and 'Transaction' on the right. A user icon is in the top right corner. The main content area has a white background with the heading 'Sign up for Warewise'. Below this are four input fields: the first three contain the text 'test', and the fourth contains '***'. A blue 'Register' button is positioned below the fields. At the bottom, a link reads 'Already have an account? [Log in here](#)'.

- User creates an account by entering account information (Name, Username, Address, Password). If all the fields are all entered correctly and the username has not been taken, the account will be created.

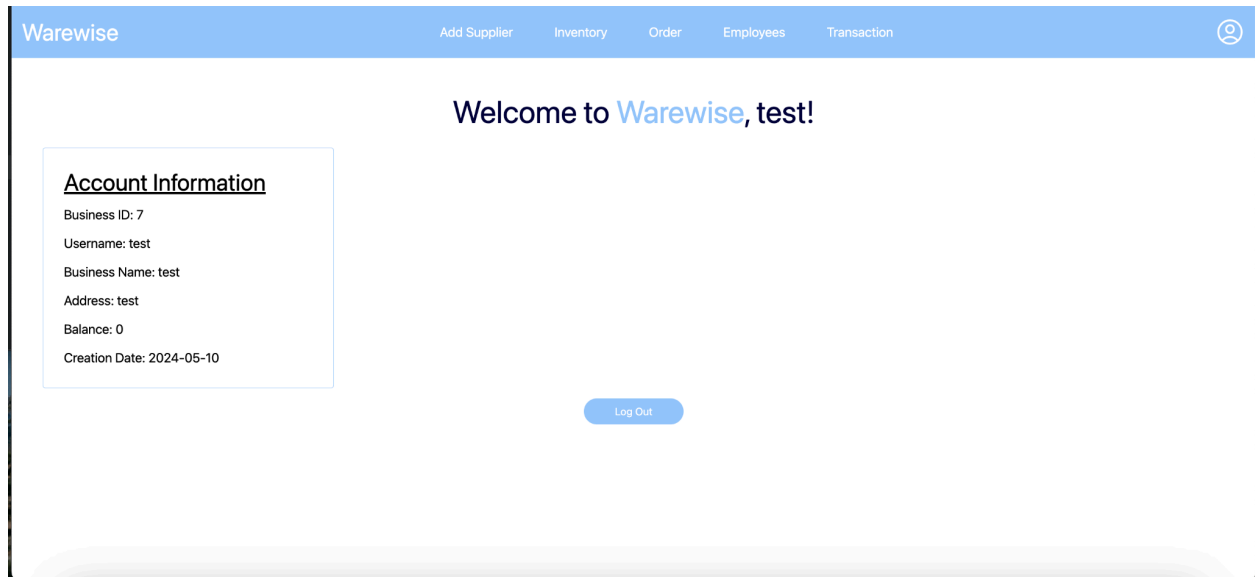
Login Page



The screenshot shows the 'Warewise' login page. At the top is a dark red browser window with tabs for 'Gmail', 'Dashboard', 'YouTube', 'Netflix', 'Google Docs', and 'HB'. A notification box from 'localhost:3000' is displayed, stating 'Logged in successfully' with an 'OK' button. The main content area has a white background with the heading 'Login to Warewise'. Below this are two input fields: the first contains 'test' and the second contains '****'. A blue 'Login' button is positioned below the fields. At the bottom, a link reads 'Don't have an account? [Sign up here](#)'.

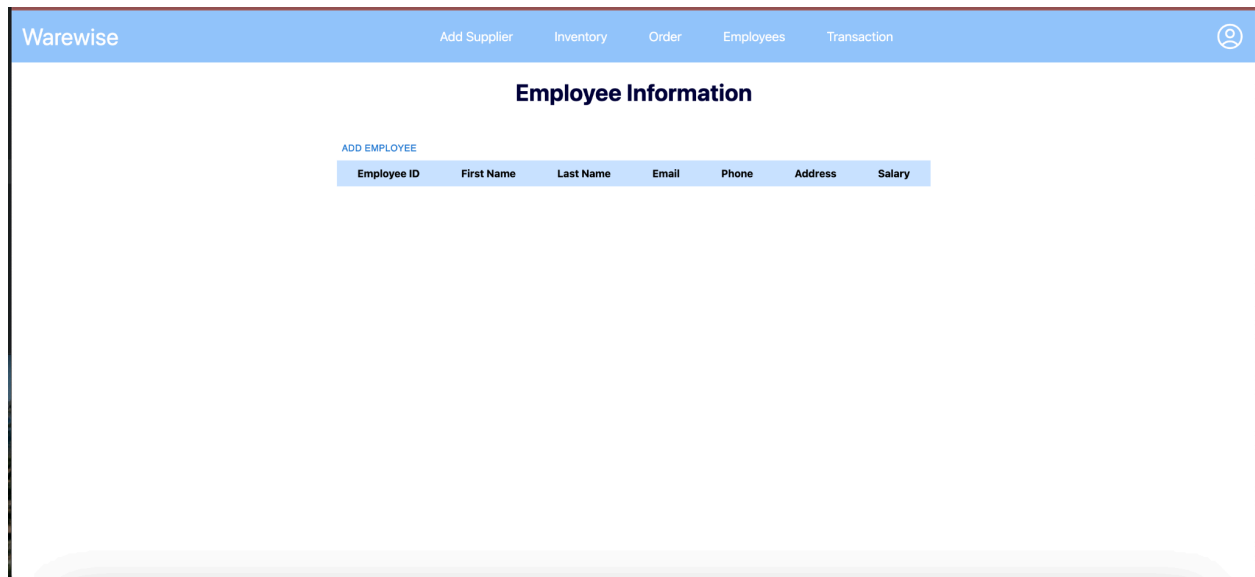
- User logs in with their username and password. If the username and password they entered are correct, an alert message will appear saying login was successful.

Account Page

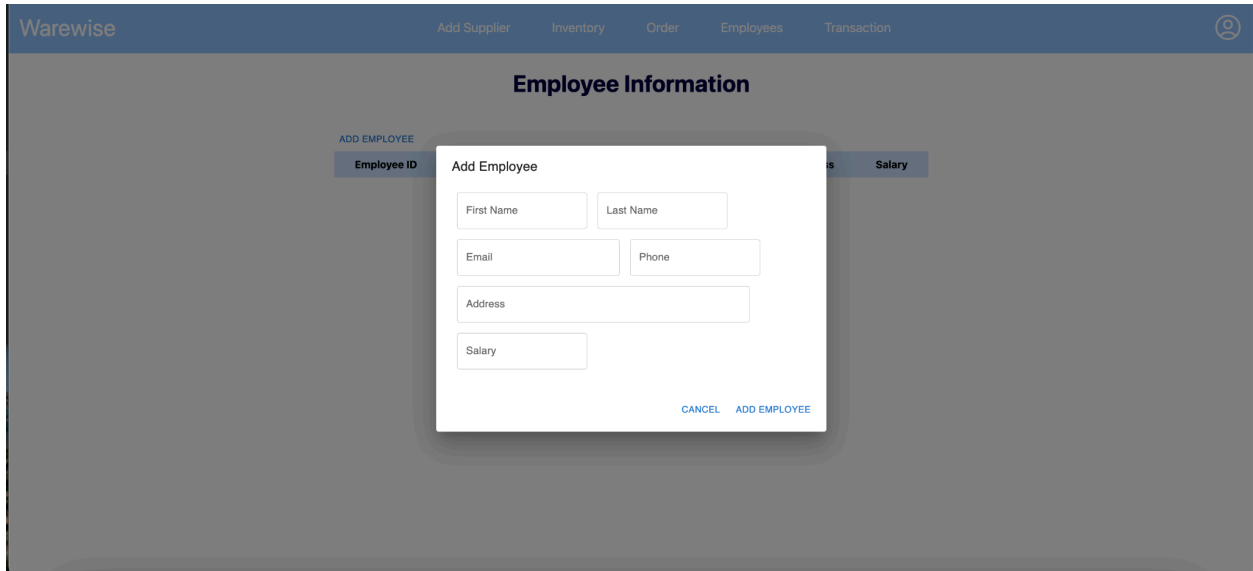


- The user gets directed to their account page where they can see their account information and also log out.

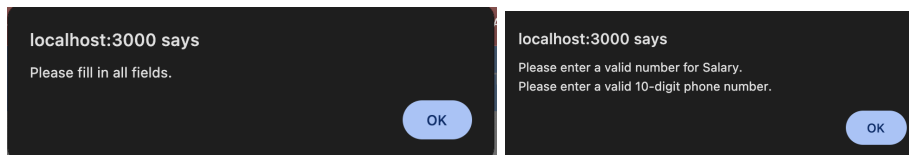
Employee Page



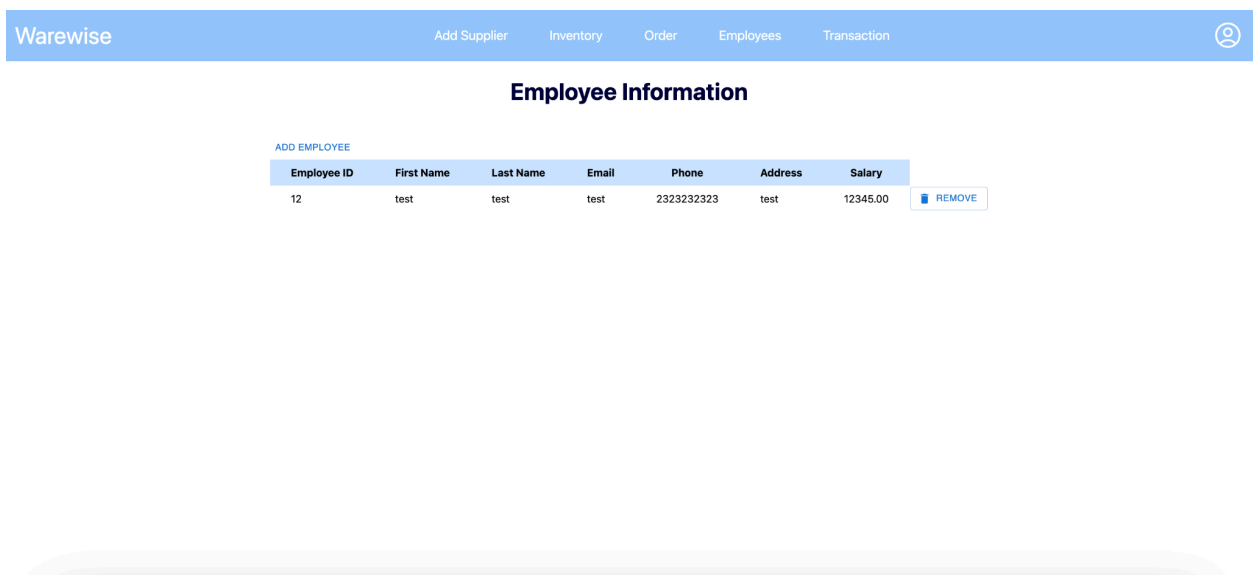
- When the user clicks on the employees tab, they can view/add/delete employees.



- If the user clicks on the add employee button, they will be prompted to enter the employee's information.

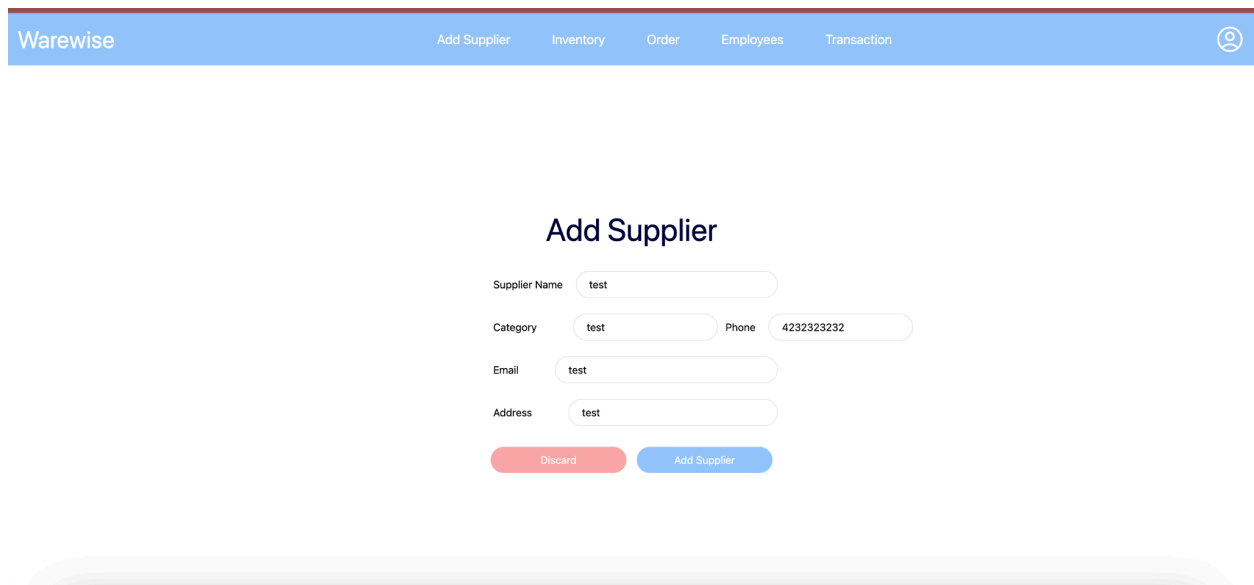


- If the user does not enter all fields, does not enter the correct format for phone number or salary, an error message will appear.



- After entering the correct employee information, the employee will show up in the table.
- The user can also remove the employee by clicking on the remove button beside the entry.

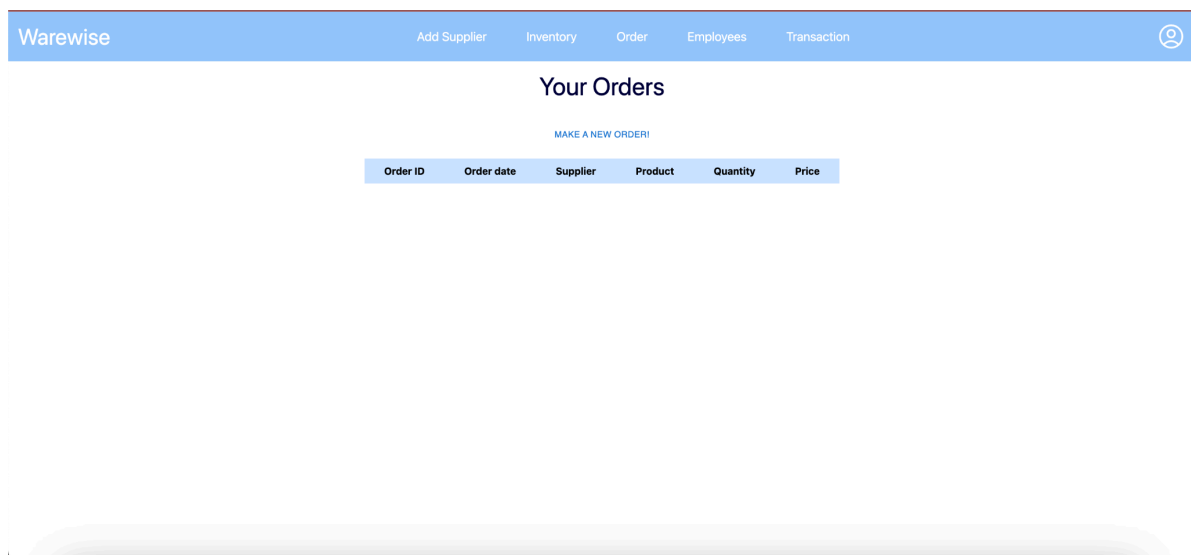
Supplier Page



The screenshot shows the 'Add Supplier' form within the Warewise application. The header bar is blue with the 'Warewise' logo on the left and navigation links for 'Add Supplier', 'Inventory', 'Order', 'Employees', and 'Transaction' in the center. A user profile icon is on the right. The main content area is white and features the title 'Add Supplier' in bold. Below the title is a form with five input fields: 'Supplier Name' (containing 'test'), 'Category' (containing 'test'), 'Phone' (containing '4232323232'), 'Email' (containing 'test'), and 'Address' (containing 'test'). At the bottom of the form are two buttons: a red 'Discard' button and a blue 'Add Supplier' button.

- If the user clicks on the “Add Supplier” tab, they will be prompted to enter the supplier information and the category of products they would like to order.

Orders Page



The screenshot shows the 'Your Orders' page within the Warewise application. The header bar is blue with the 'Warewise' logo on the left and navigation links for 'Add Supplier', 'Inventory', 'Order', 'Employees', and 'Transaction' in the center. A user profile icon is on the right. The main content area is white and features the title 'Your Orders' in bold. Below the title is a link that says 'MAKE A NEW ORDER!'. Below this link is a table with the following columns: 'Order ID', 'Order date', 'Supplier', 'Product', 'Quantity', and 'Price'.

Order ID	Order date	Supplier	Product	Quantity	Price
----------	------------	----------	---------	----------	-------

- When they switch to the “Orders” tab, they will be able to view or create orders of products.

Warewise

Add Supplier Inventory Order Employees Transaction

Your Orders

MAKE A NEW ORDER!

Order

Supplier

test

ADD

Category

Description

Quantity

Price

CANCEL PLACE ORDER

- When the user clicks on “Make new order” button, the supplier that they previously added in the “Add Supplier” tab will appear in the dropdown button. The user can then enter the rest of the information to place the order.

Warewise

Add Supplier Inventory Order Employees Transaction

Your Orders

MAKE A NEW ORDER!

Order ID	Order date	Supplier	Product	Quantity	Price
4	2024-05-10T20:33:22.000Z	test	test	5	5.00
3	2024-05-10T05:27:07.000Z	test	test4	1	1.00
3	2024-05-10T05:27:07.000Z	test	test3	1	1.00
2	2024-05-10T05:26:43.000Z	test	test2	2	2.00
1	2024-05-10T05:26:23.000Z	test	test	1	1.00

- After placing the order, the order will appear in the “Your Orders” table.

Inventory Page

Inventory

Select Category ▼

dsfd

Product Name	Product ID	Description	Quantity	Price
dfs	7	sdf	3	3.00

test

Product Name	Product ID	Description	Quantity	Price
product2	8	test	3	5.00

- In the inventory tab, the user can view all their products.

Inventory

Select Category
test ▼

test

Product Name	Product ID	Description	Quantity	Price
product2	8	test	3	5.00

- The user can also sort the products by category by using the drop down category button.

Transactions Page

Warewise

Add SupplierInventoryOrderEmployeesTransaction

Customer Transactions

Product ID

7

Quantity

2

First Name

test

Last Name

test

Email

test

Phone

3232323232

Address

test

Payment Method

test

Submit

Last Transaction Recorded:

No transactions recorded yet.

- The user can also track sales by switching to the transaction tab.
- If the user does not enter all fields or does not enter the correct format or data type for phone number or quantity, they will receive an error message.



Customer Transactions

Product ID

Quantity

First Name Last Name

Email Phone

Address Payment Method

Submit

Last Transaction Recorded:

Product ID	Quantity	First Name	Last Name	Email	Phone	Address	Payment Method
7	2	test	test	test	3232323232	test	test

- The most recent transaction will be recorded in the table.

Account Page (Updated Balance)



Welcome to Warewise, test!

Account Information

Business ID: 7
Username: test
Business Name: test
Address: test
Balance: 6
Creation Date: 2024-05-10

Log Out

- Since a sale was made, the user can see the updated balance in the account page by clicking on the profile icon.

Conclusion/Future Plans

The development of Warewise has provided us with valuable experience in not only database management systems but also frontend and backend development, as well as project management and teamwork strategies.

Possible Improvements

There are several areas where Warewise could have been improved to further enhance its functionality and usability.

Since one of Warewise's main features is ordering products from suppliers and adding to the business's inventory, role-based access control for a supplier and business account would have enabled a more streamlined interaction within our application. This feature would allow a supplier account to add and update their inventory as well as view incoming orders from businesses. In the business account, the user would be able to directly order products from the supplier and any changes in terms of inventory would be reflected in real-time between the business and supplier accounts.

Currently, Warewise only allows for one type of product per transaction. In more realistic sale scenarios, there would be more than one product per sale. As a result, we could have enhanced the transaction functionality to handle multiple products in a single transaction.

One feature we would have liked to include is an automated restocking system. For this feature to work, we would predefine a quantity threshold level for each product. When the stock of a particular product falls below its designated threshold level, the system will automatically

generate an order to the designated supplier for restocking. This feature would eliminate manual intervention in restocking products and prevent the risk of stockouts, thus resulting in less overhead and greater operational efficiency.

Lesson Learned

During the development of Warewise, one of the key lessons learned was learning how to effectively use MySQL, which played a crucial role in our project's database management. From conceptualization to query composition and execution, to identifying entities and relationships, creating tables, applying normalization techniques, and ensuring data integrity, this pivotal stage in our project development process laid a solid foundation for Warewise's functionality.

In addition to MySQL, our journey with backend development led us to explore the capabilities of Express.js, Axios, and Node.js. Leveraging Express.js simplified the implementation of server-side logic and API creation, streamlining the development of Warewise's backend infrastructure. By structuring our backend codebase with Express.js, we were able to modularize our application's functionality, making it easier to maintain and scale as our project evolved. Learning how to integrate Axios enabled us to enhance communication between frontend and backend, streamlining data exchange and improving the application's responsiveness.

Beyond the technicality of Warewise, learning the importance of thorough planning and communication among team members was significant. Being communicative and understanding each individual's responsibilities and tasks was crucial in ensuring everyone remained aligned with project goals and timelines. Being able to adapt in response to challenges and changes throughout the project lifecycle taught us how to adjust our approaches in real-time to overcome obstacles and keep the project on track.

Overall, the process of creating and developing Warewise provided valuable insights into project management, design techniques, software development methodologies, and teamwork dynamics.